

HIDING IN PLAIN SIGHT: HANDWRITING AND APPLICATIONS TO STEGANOGRAPHY

by

James Hahn

B.Phil. Computer Science, University of Pittsburgh, 2019

Submitted to the Faculty of
the Department of Computer Science in partial fulfillment
of the requirements for the degree of
Bachelor of Philosophy

University of Pittsburgh

2019

UNIVERSITY OF PITTSBURGH
COMPUTER SCIENCE DEPARTMENT

This thesis was presented

by

James Hahn

It was defended on

April 5th 2019

and approved by

Dr. Adriana Kovashka, Computer Science Department

Dr. Adam Lee, Computer Science Department

Dr. Milos Hauskrecht, Computer Science Department

Dr. Katerina Fragkiadaki, Machine Learning Department (Carnegie Mellon University)

Thesis Director: Dr. Adriana Kovashka, Computer Science Department

TABLE OF CONTENTS

| | |
|--|----|
| PREFACE | v |
| 1.0 BACKGROUND AND PROBLEM | 1 |
| 1.1 Problem Statement | 1 |
| 1.2 Outline | 4 |
| 1.3 Definitions | 5 |
| 2.0 RELATED WORK | 7 |
| 2.1 Classical Methods | 7 |
| 2.2 Machine Learning Methods | 10 |
| 2.3 Steganalysis Tools | 13 |
| 3.0 HANDWRITING GENERATION | 14 |
| 3.1 Recurrent Neural Networks | 14 |
| 3.2 Related Work | 15 |
| 3.3 Dataset | 18 |
| 3.4 Results | 19 |
| 4.0 DATA ENCRYPTION | 23 |
| 4.1 Traditional Methods | 24 |
| 4.2 Neural Cryptography | 25 |
| 4.3 Dataset | 29 |
| 4.4 Methods | 29 |
| 4.5 Results | 32 |
| 5.0 STEGANOGRAPHY | 35 |
| 5.1 Historical Background of Steganography | 35 |

| | |
|---|-----------|
| 5.2 Neural Steganography | 39 |
| 5.3 Image vs. Non-image Data | 41 |
| 5.4 Methods | 41 |
| 5.5 Results | 43 |
| 6.0 ENCODING & DECODING DATA | 46 |
| 6.1 Pipeline & Integration | 46 |
| 6.2 Recognizing & Decoding the Sequence | 48 |
| 6.3 Results | 51 |
| 7.0 CONCLUSIONS | 55 |
| 7.1 Discussion | 55 |
| 7.2 Limitations | 56 |
| 7.3 Problem Generalizability | 57 |
| 7.4 Future Work | 57 |
| BIBLIOGRAPHY | 62 |

PREFACE

Acknowledgements to the University of Pittsburgh Computer Science Department and University Honors College for providing necessary resources throughout my undergraduate career to develop the necessary leadership, intellectual, and technical skills to undergo this monumental project. They single-handedly shaped my career and future by providing countless opportunities not offered at many universities, which I owe gratitude toward.

Special and insurmountable thanks are given to my research advisor, Dr. Adriana Kovashka. She provided my first chance to explore the world of computer vision, both academically and within the research community. She provided countless letters of recommendation throughout the several semesters we have known each other (and probably too many). I took her graduate computer vision course and her machine learning course, then she offered the chance to co-organize a workshop at CVPR 2018, and finally guided me toward completion of this thesis. There are not enough words to explain my appreciation for all she provided through career, academic, and life advice.

Finally, this thesis is dedicated to my loving parents, Carol and Steve. Despite not being given the same educational opportunities as I, they always encouraged me to excel in school. I was never the best child growing up, so a lot of credit is owed to them for raising me with high morals, providing the freedom to explore my interests, and for always supporting me.

HIDING IN PLAIN SIGHT: HANDWRITING AND APPLICATIONS TO STEGANOGRAPHY

James Hahn, B.Phil

University of Pittsburgh, 2019

Steganography, the process of hiding secret information within normal-looking data (a cover), lies at the intersection of computer science and security. Recently, these covers have come in the form of images. The steganographic task must pass two visual inspections: by humans and by machines. If a human examines the image and fails to spot the hidden data, the algorithm successfully passes the first test. The second task is preventing a machine from detecting patterns to reverse engineer the secret information. In 2017, researchers achieved some success [1], but there were two main issues: the steganography only worked with (a) fully saturated, (b) fixed-size (100×100) images. To curb these limitations, a new pipeline is explored to generate non-fixed size cover images with steganographic *modification* rather than *embedding*. This paper explores this new form of steganography with several key processes. First, the secret information is encrypted before combining it with the cover using neural cryptography. Second, the information hides in the stroke data of a person's handwriting on a white background, increasing task difficulty, forcing the steganographic approach to be robust to a plethora of data, including sparse images. In this sense, the strokes are directly modified, rather than inserted or embedded in-between pixels. The result is a toy problem utilizing realistic, generated, coordinate sequences of human handwriting modified with slight offsets dependent on the information combined with the coordinates in the sequence. With these slight offsets, the new generated coordinates are nearly identical to the original coordinates, preserving the primary structure of the handwriting, but shining light on a new avenue of steganography based on data *modification* rather than *embedding*.

1.0 BACKGROUND AND PROBLEM

First, in order to understand the significance of this research and project, one must be aware of the state-of-the-art in steganography, its shortcomings, and a plan to improve it. We propose a toy problem, orthogonal to current approaches, with potential to improve the current state-of-the-art. Hopes are the toy problem can be abstracted into more general case studies. First, steganography is introduced. Then, potential issues are discussed, a toy problem is introduced, its step-by-step solution is proposed, and an outline is provided in order to guide the reader from early steps of information hiding all the way to decoding the secret message.

1.1 PROBLEM STATEMENT

The field of steganography is a historied sub-field of security, but has recently intersected with machine learning. Recently, popular applications arise in hiding data in images with computer vision. The field revolves around the idea of inserting a secret message, or secret data, into normal-looking data. When applied to images, this comes in the form of inserting image A into image B with the output looking as close to image B as possible. In this scenario, image A is the secret image/data and image B is the cover image. Their output, image C, is referred to as a container. A more concrete example is using an image of a cat as the cover and an image of an elephant as the secret data. When combined, the goal is to output an image with appearance as close to the original cat picture as possible.

This idea can be applied to data in general, not just images, including audio and sensor data. The field has been revived recently by [1]. Their goal is to implement the exact

strategy described above. While notable results are achieved, several issues are identified. The authors experiment with 100×100 pixel images. While medium-sized, the image sizes are not easily scalable if the data is of larger or smaller size. As such, if the goal is to hide a small 10×10 image, two problems arise:

1. Upsampling is required to increase the secret image from 10×10 to 100×100
2. Resources are wasted on both upsampling and transferring such small data

Not only are extra operations required to manipulate the small data, but the actual information may lose small, local features once upsampled, risking data loss when decrypted, or extracted, from the container image.

Conversely, if the secret image/message is larger than 100×100 pixels, the data must be broken up into smaller chunks and “steganographized” with multiple images. This produces clear problems, such as higher risk of data loss after decoding, the transfer of several container images across networks (a natural process and major motivation when communicating secret information), and a significant amount of extra time to process the new containers. Additionally, if the data is not a multiple of 100 in either width or height, the leftover chunk must be combined with zero-valued pixels to ensure it is 100×100 . Therefore, one solution is to re-train a new network with higher dimensionality (i.e., 1000×1000) to embed more data or train a smaller network (i.e., 10×10), but only provides a short-sighted remedy for the larger issue at hand, which is embedding secret data of different sizes in cover data of different sizes efficiently without significant modification to the process or pipeline.

Additionally, most of these images are fully saturated, RGB images. As such, a 100×100 image contains $100 \times 100 \times 3 \times 8 = 240,000$ bits of data. While this allows for much freedom in data transmission, allowing a significant number of bits for embedding, the current state-of-the-art generally fails to work on sparse images (observed in Figure 2 for a blank cover), or images with significant chunks of whitespace. This produces collateral problems in the case where the data is audio or non-image data. With images, this sparsity issue will pass human inspection but fail machine inspection. However, when not using images, the sparsity will cause failures in both inspections.

Finally, the data is not encrypted. This lack of added security means any steganalysis

tools capable of cracking the steganography algorithms will instantly gain access to the data. However, if the data is encrypted before being combined with the cover image, an extra level of security is added to the steganography. To achieve this, neural cryptography, a relatively new field at the intersection of cryptography and machine learning, is explored to generate unique encryption schemes.

As such, state-of-the-art in steganography produces substantial results, but has faults in three main areas: fixed data size, dealing with sparsity, and lack of encryption. To solve these multifaceted issues, we propose a novel solution: human handwriting is utilized to produce free-form, unfixed size images capable of avoiding sparsity issues in images, resulting in an approach with the ability to generalize to non-image data.

First, secret data M (e.g., “The password is 123”) is input by a user. Next, the secret data is encrypted to produce new data M' . Then, the approach generates coordinates for a person’s handwriting strokes with a random message (e.g., “The sky is blue”) which would be plotted onto an image. For a given written character, 20-50 coordinates C are produced. Then, each coordinate is combined with secret data M' to produce a new coordinate C' with nearly identical x and y values. These coordinates C' are plotted on an image to produce realistic handwriting. To decode the secret message, the output image is read in, each coordinate C' is recognized, C and M' are extracted by decrypting C' , and the process of encryption is reversed on M' to produce the original message M . This solves the issue of encryption since we encrypt the data as one of the first steps. Also, the secret data can be any size desirable and it will fit into an image which can be as long as required to fit the secret data. Third, images of handwriting are generally sparse since handwriting is usually placed on a white background with black text, so the limitation to dense images is alleviated. Finally, the decoding task for an adversary is generally more difficult than the steganography algorithm proposed by [1], since the latter utilizes fixed size inputs and outputs and an adversary must only seek out the correct weights to decode the message given the container image. Meanwhile, the proposed approach requires an adversary to recognize all 20-50 coordinates per handwritten character in an unsupervised manner, then learn the additional weights of the machine learning models, requiring an extra step along the way. Additionally, to decode, adversaries must segment individual pixels of the image, whereas [1]

just requires adversaries to train on entire images to decode, which is more straightforward.

The goal of this research is to construct a pipeline with several levels of security capable of being deployed across open data channels without suspicion of secret information being contained. A forward, encoding pass is constructed, and a subsequent decoding pass is constructed. Analysis of the security strengths and potential weaknesses will be analyzed, as well as the generalizability. With this pipeline, we hypothesize that information can be secured by steganographically embedding a message into a sparse cover image through modifying the appearance of the image, rather than the contents of the image.

1.2 OUTLINE

This research requires integrating several steps. First, it requires the processes to generate human-like handwriting. Then, a message must be retrieved from the user and encrypted to provide an added level of security in the future pipeline. Finally, the encrypted data must be emplaced into the generated handwriting to fix sparsity issues with current steganography approaches and conceal the fact any secret message exists. Finally, this encoding process is complex enough that the decoding process requires an entirely new pipeline, requiring more work on the engineering end, but it is important to note this requires extra thought for adversaries when attempting to decode the secret message, marginally supplementing the existing secure methods in the pipeline. This decoding process is unseen in prior steganography approaches, thus adding to the novelty and adding another barrier of security against adversarial attacks. The pipeline discussed above is broken into three main processes for encoding, and one large reverse engineered pipeline for decoding data. As such, this document contains discussion of the following, in order:

1. Exploring several avenues to generate handwriting
 - Applicability of machine learning models
 - Applicability to the toy problem
2. Data encryption
 - Current encryption state-of-the-art

- Recent intersection of machine learning and encryption
3. Steganographizing data
 - Modifying appearance and displacements of handwritten strokes
 - Difficulties, embedding ratios, and quality of results
 4. Decoding the data
 - Recognizing the coordinate sequence's beginning
 - Predicting sequential coordinates
 - Extracting encrypted data from steganographized coordinates
 - Decrypting the retrieved data
 - Engineering feats and real-world applicability
 5. Future work
 - How can this toy problem be extended to larger problems?
 - How can this approach be improved in terms of quality and quantity?
 - Where can this work be used where it has not been used before?

1.3 DEFINITIONS

Finally, although not much jargon exists for steganography, there are a few key words worth noting. In addition, since this project requires the integration of three key processes integrated into one pipeline spanning three areas of computer science and security, terminology can easily be thrown around and it quickly becomes confusing. Therefore, important words and their definitions are supplied below:

- **Cover:** An arbitrary piece of data that seems normal to a human or machine, but is eventually combined with a series of secret data to produce container data.
- **Container:** A combined form of data containing information of the cover data and the secret data. This should represent the cover data as closely as possible.
- **Key:** Commonly used in encryption as a form of hashing data.

- **Adversary:** An attacker attempting to extract secret data from a container without access to a key or prior knowledge of the true, underlying extraction techniques and algorithms.
- **Steganalysis:** The process of analyzing a container to detect the presence of steganographed data or the actual contents of the secret data.
- **LSB:** A common and foundational algorithm in steganography where bits of the secret data are placed in the one, two, or three Least Significant Bits of the cover data.
- **Cryptography:** The act of concealing secret information even if adversaries know it has been transformed.
- **Steganography:** The act of concealing secret data in normal-looking data. The goal is to hide the existence of a secret message.
- **De-stenographize:** The act of extracting secret data from a container.
- **Alice:** A common name in security given to the agent encoding/encrypting data.
- **Bob:** A common name in security given to the intended recipient of encrypted data with the goal of decrypting the secret data.
- **Eve:** A common name in security given to an eavesdropper. In this paper, the machine learning algorithms are trained to increase this agent's decryption error.
- **Dave:** A name given to an eavesdropper. This agent differs from Eve where the algorithms are not trained to be robust to his algorithm/model.
- **Ciphertext:** Another term for encrypted data.

2.0 RELATED WORK

Due to its vast history, a plethora of algorithms have been developed for steganographic purposes. With that being said, the research and developments can be broken into two areas: classical methods and machine learning methods. Classical methods include the original forms of steganography, such as etching messages into a slave’s head and forcing them to grow their hair [2]. However, for the sake of this paper’s modernity, it refers to modern computational methods modifying the metadata of a media file, randomized methods, and the least significant bit (LSB) algorithm [3]. Meanwhile, with the rise of computation power and usage of neural networks, automated algorithms have emerged by training against adversarial algorithms. Specifically, for image steganography, the rise of convolutional neural networks (CNNs) and Krizhevsky et al’s “AlexNet” in the 2012 ImageNet Challenge resulted in widespread use of CNNs [4]. Finally, to combat the development of robust steganographic algorithms, researchers in the field of steganalysis, or detecting the presence of secret information, have developed a variety of tools to break these algorithms.

2.1 CLASSICAL METHODS

Steganography is the method of hiding data in secure communication channels such that existence of a secret message is not apparent, therefore not drawing attention of adversaries, or eavesdroppers. The word has roots with *steganos* meaning “covered writing” [5] and *graphien* meaning “to write” [6], producing the combined definition of “hidden writing” . The field itself, with countless applications, is broad. There are many varieties of algorithms and approaches to hiding data. A plethora of algorithms exist for hiding specific types of data.

The field has historical significance dating back to 440 B.C. [2] with simple, physical steganographic algorithms, such as etching messages onto a messenger’s head, disappearing ink, and verbal/visual/physical morse code. The field took a turn around the 20th century with the invention and power of modern computers, creating digital steganography. Within digital steganography, three main research areas exist: steganography with images, steganography with audio, and steganography with text.

As previously mentioned, the most common and widely used steganography algorithm is LSB, or Least Significant Bit. The origins of this algorithm are unknown, but one of the oldest mentions of the algorithm in digital steganography is found in a paper by Bender et al. in 1995 [3]. With this simple algorithm, secret data is broken into sequential chunks of 2 bits. Then, each chunk of 2 bits replace the 2 least significant bits of each 8-bit pixel channel (red, green, and blue) in an image; this simple approach can easily be abstracted to other forms of data besides images. The general idea is an 8-bit pixel reading the value 253 might be changed from 11111101 in binary to one of: 11111100 (decimal: 252), 11111101 (decimal: 253), 11111110 (decimal: 254), or 11111111 (decimal: 255). In contrast, if the two most significant bits (the bits all the way to the left) were changed, the value of the pixel would change significantly. For example, if one attempted to encode the secret data **01** in binary into the pixel value 11111111 (decimal: 255) by changing the 2 most significant bits, the value changes from 255 to 127 (11111111 \rightarrow 01111111), which the average adversary would easily detect. More on this method, as well as a real-world example, is discussed in Section 5.1.

Variations of the LSB algorithm exist, but the same general issues persist. On an important note, a paper by Sugandhi et al. in 2016 encrypted the data and embedded the new data into the least significant bits [7]. Although another level of security was added, making the algorithm more robust, the approach still has issues with low embedding ratios. In this algorithm and future algorithms, an embedding ratio is described as dividing the number of bits of the secret data by the total number of bits in the cover data. For example, a secret image of size $N \times N$ has N^2 bits. Similarly, a cover image of the same size has N^2 bits, resulting in an embedding ratio of $N^2/N^2 = 1.0 = 100\%$. Using a higher order than 3-LSB will most likely cause issues and a human adversary should easily be able to detect

presence of random noise. Therefore, the approach is robust against adversaries retrieving the secret information, but not so much against adversaries detecting the presence of secret information. Moreso, if AES [8] or RSA [9] are utilized to encrypt data, the transmission of a key to the receiver must be securely sent. So, if the key is intercepted by an adversary, the problem simply turns into LSB and nearly all security is lost.

Another approach utilizes DCT (Discrete Cosine Transform) to modify images. This algorithm, although popular in steganography, is commonly used in JPEG compression due to its ability to separate an image into “high frequency”, or high importance, and “low frequency”, or low importance regions [10]. The same general concept is used in this algorithm, where coefficients are calculated for each pixel and information is placed into higher priority pixels, which typically have a higher intensity. For an image with height of N and width of M , a series of coefficients are computed for every pixel with the following equation:

$$C(u, v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cos \left[\frac{(2x+1)u\pi}{2N} \right] \cos \left[\frac{(2y+1)v\pi}{2N} \right] \quad (2.1)$$

where $C(u, v)$ is the DCT coefficient for the pixel in row u , column v and $f(x, y)$ is intensity of the pixel in row x , column y . This algorithm is typically referred to as a frequency domain approach, contrasting with LSB, which operates in the spatial domain [11]. Typically, an image is divided into 8×8 pixel blocks in practice, replacing N and M above with 8. Then, DCT coefficients are calculated over each pixel and the information is dispersed to each, forming a DCT coefficient matrix. Then, the pixels with higher intensities, or “higher frequencies”, when modified, typically go unnoticed compared to a pixel with lower intensity. So, the information is laid out to modify pixels in order from highest intensity to lowest intensity [12]. Finally, to extract the information from the pixel, an Inverse Cosine Transform can be used by simply solving for $f(x, y)$ and subtracting the necessary information for each 8×8 (in general, $N \times N$) block. While useful, this algorithm typically has a low embedding ratio and is easy to detect with utilization of the Inverse Cosine Transform.

2.2 MACHINE LEARNING METHODS

As mentioned earlier, 2012 marked a turning point in the vision and image processing community with the massive success of AlexNet in the ImageNet Challenge [4]. A few steganography algorithms, such as LSB, exploited spatial structure of images, but convolutional neural networks allowed full utilization of an image’s spatial properties through learning filters, and thus high level properties of the image. As such, CNNs in steganography allow a machine to recognize patterns and thus embed secret patterns in an image.

In 2017, [1] proposed a modern machine learning steganography method. The entire process reads a secret, $N \times N$ RGB image S and passes it through a prep network, separating the red, green, and blue channels of the image. Then, an arbitrary $N \times N$ cover image C is randomly chosen from the ImageNet database containing 14,197,122 images [13, 14]. A hiding network, which is a neural network with $N * N$ input neurons representing the N^2 pixels and their three RGB channels, combines S and C into an $N \times N$ container image C' . This hiding network consists of five convolutional layers, with 50 filters each of dimensions 3×3 , 4×4 , and 5×5 , resulting in 150 total filters per layer. Finally, once the secret data is hidden, in order to decode it, a reveal neural network is constructed. The authors mention an autoencoder structure [15] to relate the hiding and reveal network, but do not explicitly mention the architecture for the reveal network. Therefore, to fill in gaps in [1], we assume the reveal network resembles a near-identical structure to the hiding network. The autoencoder is trained to optimize a loss function combining decryption error and cover error:

$$\mathcal{L}(C, C', S, S') = \|C - C'\| + \beta \|S - S'\| \quad (2.2)$$

where the first term represents the L_2 error between the container C' and C (i.e., cover error), the second term represents the L_2 error between the true secret message and the extracted secret message from the reveal network (i.e., decryption error or reconstruction error), and β is some constant weight on the decryption error term.

This entire process is illustrated in Figure 1. Results were notable because they did not

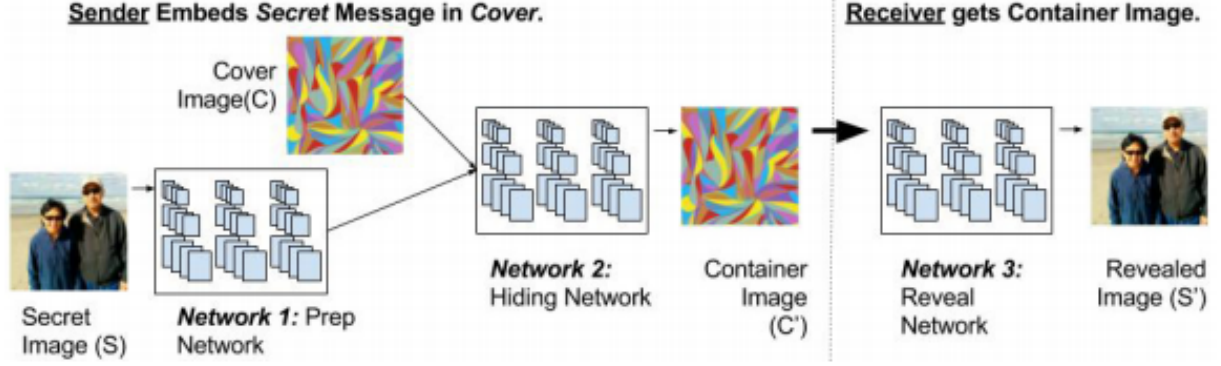


Figure 1: Convolutional Neural Network approach to steganography, produced by [1]

end up learning the LSB algorithm, but rather its own unique algorithm. With $\beta = 1$, the best balance of cover error and decryption error was achieved. For each channel on each pixel, the cover error was about 3 intensity values (on a scale from 1 to 256), where the decryption error per channel per pixel was 3.2 intensity values. When β was reduced, cover error decreased, but decryption error increased, and vice versa when β increased. As a special case, when $\beta = 0$, or when the loss function consists only of the cover error, the cover error was 0.1 intensity values per channel per pixel. Even more impressive, the embedding ratio is 100%, which far surpasses classical methods, such as LSB, which commonly only achieves a 25% ratio. Despite its success, there are several faults. The networks are trained on full RGB images, the network images must be of fixed size, and the secret image was not encrypted beforehand. Also, the network was not trained to be robust against a neural adversary, which is common in security research applications, so no steganalysis was performed in the paper. If the cover image is a white background, or contains empty regions, the networks generally perform poorly in terms of either cover error or decryption error, or sometimes both. As such, dealing with sparsity is an issue. Finally, for an adversary to break the hiding network, training on image pairs of ImageNet images and the hiding network's output should allow a machine to recognize patterns and decode the secret message. Some results of special cases can be seen in Figure 2.







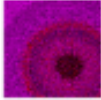

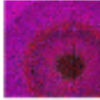

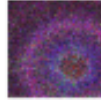

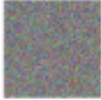




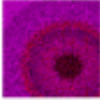









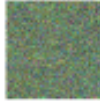

| | Original Images Cover / Secret | | Reconstructed Img Cover / Secret | | Residual Error Cover / Secret | | |
|--|---|---|---|---|---|---|---|
| Blank Cover Colorful Secret | |  |  |  |  |  | Pixel Error: 6.0 / 8.8 |
| Natural Cover Artificial Secret |  |  |  |  |  |  | 8.3 / 28.8 |
| Natural Cover Random Secret |  |  |  |  |  |  | 7.1 / 45.1 |
| Artificial Cover Natural Secret |  |  |  |  |  |  | 28.9 / 29.6 |
| Random Cover Natural Secret |  |  |  |  |  |  | 39.4 / 17.2 |

Figure 2: Special cases of cover and secret images from [1]

To the best of my knowledge, no other notable machine learning approaches to steganography exist. Digital steganography, especially utilizing machine learning techniques, rose in advancements with [1]. This research, although utilizing machine learning, differs significantly in approach, embedding, and training compared to every mainstream steganography approach. For example, generated handwriting can produce an image of any size, information is embedded into the coordinate data of the handwriting strokes rather than individualized pixel value, therefore modifying the high-level properties of the image. Finally, secret data is encrypted before being steganographed with handwriting coordinates, which is a novel addition to steganography techniques.

2.3 STEGANALYSIS TOOLS

Steganalysis is a natural response to the field of steganography. Much of the research advances alongside the state-of-the-art in steganography with the primary goal to detect presence of secret information as well as decoding the secret data. Many steganalysis tools exist. Several were mentioned above for the popular steganography algorithms, while [16] contains an agglomeration of the most popular steganalysis tools for varying algorithms.

One common steganalysis tool is StegExpose, a code repository focused around detecting LSB steganography. StegExpose is utilized in [1] to detect the presence and placement of hidden information. They find the secret information is not simply placed into the least significant bits of the data. Another approach utilized in the paper is flipping bit values across color channels in the container image. A flipped bit value impacts the distribution across all color channels for the reconstructed secret image, displaying robustness of the approach. However, this only works for secret data in the form of images as non-image data is sensitive to small fluctuations in reconstruction errors, in contrast to small offsets of pixel intensity values in the reconstructed image.

An alternative, modern approach to steganalysis is detecting through neural networks. [1] tests an additional binary classifier with negative images as ground-truth images from the ImageNet database, and positive images as real container images with secret data. The classifiers achieve anywhere from 90-99% success in detection of the steganographed image. As such, their high embedding ratio leads to high detection rate by neural steganalysis tools.

Meanwhile, [17] measures its success through another steganalysis tool, ATS (Artificial Training Sets) [18]. This tool produces steganographic images as a training set, trains a supervised classifier on this training set, and outputs whether a given image is steganographic in nature.

These three approaches are key in evaluating steganographic images. Further investigation into their methods and success on current steganography papers are discussed in Chapter 5. Regardless, these methods are imperative to ensuring success of current methods and are always evolving in order to detect presence of data hidden in various permutations and areas of an image.

3.0 HANDWRITING GENERATION

Most research related to handwriting is concerned with the state of handwriting recognition. In fact, the entire ICFHR (International Conference on Frontiers in Handwriting Recognition) conference is dedicated to specifically advancing the state of recognizing handwriting [19, 20, 21]. Additionally, with the invention of recurrent neural networks and generative adversarial networks, research has produced impressive results in video frame prediction [22, 23, 24], image reconstruction and synthesis [25], pose generation [26], style transfer [27], machine translation [28], visual-question answering [29], and even handwriting recognition [30, 31, 32, 33, 34]. Despite all success with handwriting recognition, handwriting generation is generally left unexplored. As such, in this pipeline, the first of three steps to encoding data is generating realistic, human-like handwriting. This can be accomplished through two different avenues: generative adversarial networks (GANs) and recurrent neural networks (RNNs).

3.1 RECURRENT NEURAL NETWORKS

The idea of recurrent neural networks has existed since Rumelhart et al’s work in 1986 [35], but was further extended into the modern perception of a machine learning algorithm for temporal data founded upon Hopfield networks and deep learning [36, 37]. With the invention of long short-term memory and gated recurrent units [38, 39], the vanishing gradient problem in traditional RNNs disappeared. GRU and LSTM units operate by adding additional pointwise operations within their cells, as seen in Figure 3. These pointwise operations allow for the addition of a previous cell state memory and operations allow for more complex

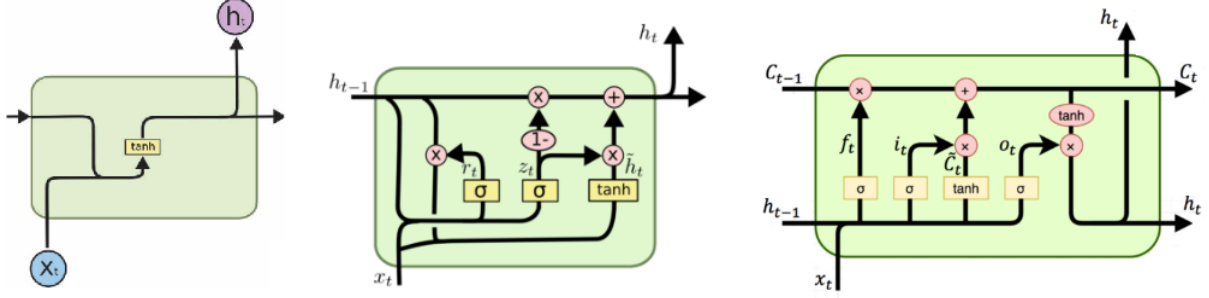


Figure 3: Left: vanilla RNN hidden unit. Middle: Gated Recurrent Unit (GRU) with hidden weights and an input vector as inputs and hidden weights and an output vector as outputs. Right: Long Short-Term Memory (LSTM) unit with a cell state, hidden weights, and input vector as outputs, with a new cell state, hidden weights, and output vector as outputs. Images taken from [41].

operations to ‘remember’ arbitrarily long sequences of data. Specifically, a GRU utilizes an update gate and reset gate to manage memory, while an LSTM utilizes a forget gate (whether to erase memory) and an output gate (how much data to reveal) to manage memory [40]. Since LSTMs contain more structure to dictate which data and how much data to pass through the network, they generally produce higher accuracies on tasks compared to vanilla GRUs and RNNs in order to accurately model arbitrarily long data sequences, making them a strong candidate for handwriting data in sequential coordinate form.

3.2 RELATED WORK

Not much research exists for handwriting generation, let alone at the intersection of handwriting generation and RNNs. In fact, a majority of code repositories centered around handwriting generation [42, 43, 44, 45, 46, 47, 48] utilize Graves’ work with RNNs and handwriting generation in 2014 [49]. The only two notable exceptions to this. A paper in 1993 focused on accounting for handwriting velocity for generation with foundations on

mathematical models [50]. As such, their methods were formed on a small dataset of 1,052 samples, the approaches are not modern, techniques are not easily implementable, and they place too much emphasis on speed of handwriting rather than pure temporal and location data, which is the premise of the IAM Online Handwriting Database (IAM-OnDB), a handwriting dataset containing coordinates of a pen in real-time, so their methods are not applicable to this research. [51] expands the IAM-OnDB with additional samples and uses a recurrent neural network and gaussian mixture model (GMM) [52] to generate sequences of coordinates. Not only do they perform the handwriting generation, but they also allow style transfer and editing of text after it has been written. Despite this, the approach is similar to [49] and there is a lack of code online. So, for the sake of time and convenience, the former will be utilized to generate handwriting. Graves' work is an ideal display of the recreation of realistic human handwriting with large variety.

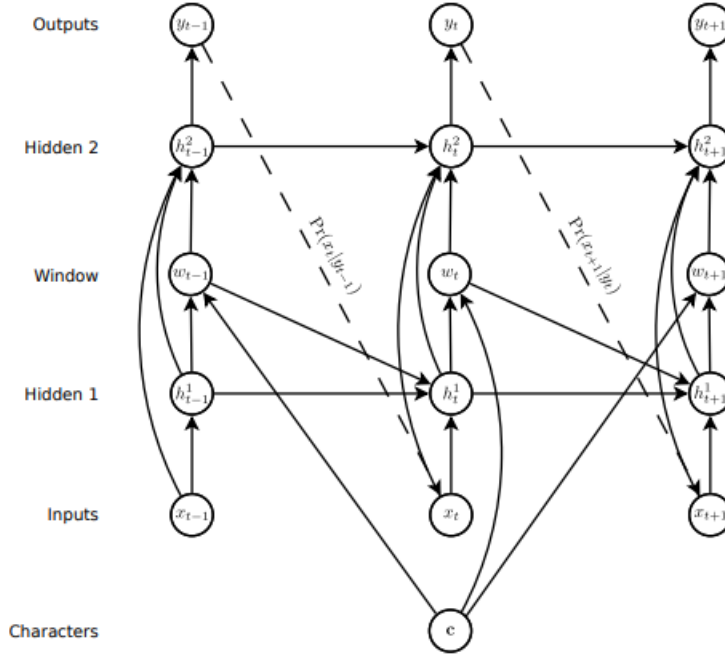


Figure 4: RNN architecture used for stroke coordinate synthesis in [49].

In his paper, Graves utilizes LSTM (Long Short-term Memory) units, a specialized form of recurrent neural networks capable of learning long-term dependencies of time series data, to generate coordinate sequences of realistic handwriting samples. First, a series of hand-

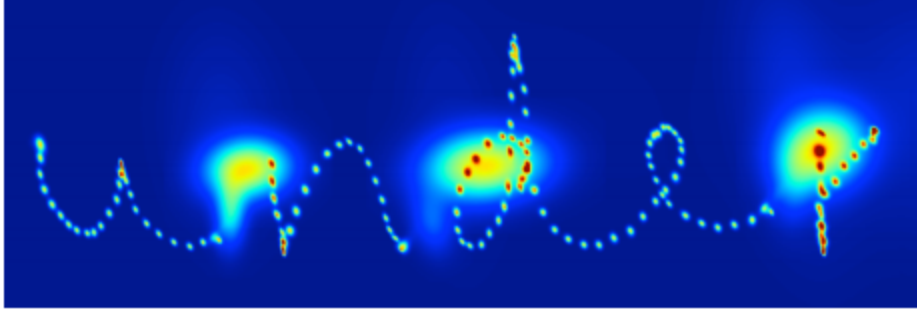


Figure 5: View of mixture density network predictions from [49].

written strokes are broken up into a training set of 5364 samples, two validation sets of 1438 and 1518 samples, and a test set of 3859 samples. Each sequence is represented as an in-order sequence of pen-tip locations from one of many different writers. Additionally, each coordinate has three values: x , y , and a 0 or 1 flag indicating the user ended a stroke and lifted their pen.

A mixture density network [53] is used to find parameters, or neural network weights, for a mixture of probability distributions. Figure 5 showcases the coordinates of the word “under” being predicted in real-time. Small blobs indicate most probable coordinates based on previous coordinates. Large blobs appear at the end of a sequence and indicate a probability distribution of the beginning of the next sequence. At every iteration, the coordinates are drawn from these distributions, some of which may be larger in area than others depending on style of authors.

One trick Graves employed to control variance in legibility of handwriting was to bias the distributions. A higher bias forces lower variance in the probability distributions. As bias approaches infinity, the distribution grows tighter and tighter until essentially the same coordinate is output for a fixed prior coordinate sequence. As such, as bias increases, so does the handwriting legibility and vice versa.

3.3 DATASET

The IAM Online Handwriting Database is viewed as the golden standard of online handwriting [54]. Online handwriting is best defined by contrasting it with offline handwriting. The most widespread offline handwriting database is the MNIST [55, 56, 57], and more recently, EMNIST (Extended MNIST) datasets [58, 59]. These datasets only contain 28×28 images of the alphanumeric characters (0-9 and a-Z), the former being a subset of the latter. Some variety exists among the characters, but they are largely the same structure and none are cursive, as seen in Figure 7, with a simple label from 0 to 61 (number of items in the RegEx set [0-9a-Z]). In contrast, the IAM dataset, seen in Figure 6 and initially published in ICDAR 2005 [60], consists of individual handwriting strokes from 221 humans writing on a whiteboard. The name is often shortened from IAM Online Handwriting Database to IAM-OnDB, which will be the case for the remaining portions of this paper. IAM-OnDB contains 86,272 word instances from an 11,059 word dictionary, providing word structure non-existent in MNIST or EMNIST. In addition, samples contain higher variance of handwriting since each character depends on previous integration with other characters, such as an ‘l’ being continuously connected with an ‘e’, altering the basic structure of both characters. This realistic property does not exist in datasets with isolated characters. Additionally, by the nature of the handwriting, non-alphanumeric characters and punctuation exist in IAM-OnDB, such as hyphens, quotes, commas, periods, and apostrophes. Finally, instead of solely labelling text images with their message contents, the coordinates, describing the sequence of strokes a human took to complete the sample, were tracked with the eBeam software [61] with the capabilities to track pen positions using infrared signals. This last bit of information is the primary distinction between offline and online handwriting datasets. If the process of handwriting is sequentially tracked, such as with coordinates, the dataset is online. If the process of handwriting is static and consists solely of images of specific letters or words, the dataset is offline. The online aspect of IAM-OnDB is important to understanding how Graves’ work differs from traditional handwriting recognition and generation.

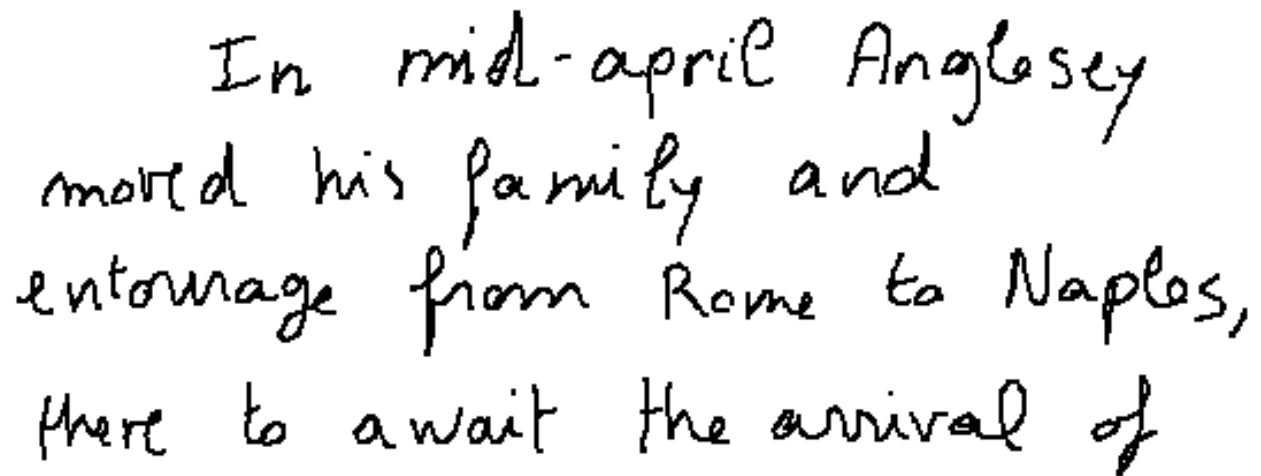
A handwritten text sample in black ink on a white background. The text is written in a cursive style and reads: "In mid-april Anglesey moved his family and entourage from Rome to Naples, there to await the arrival of". The handwriting is fluid and somewhat informal, with varying line heights and slanted letters.

Figure 6: A depiction of common items in the IAM Online Handwriting Database [54]

3.4 RESULTS

Several implementations of Graves et al's work has been produced, but [62] provides the best code readability, structure, and flexibility in terms of parameter tuning for generating handwriting samples. As such, any generated handwriting is produced with their work. Specific samples can be found in Figures 9 and 8. First, twelve styles of handwriting can be produced with the work with varying levels of width, spacing, cursive structure, and readability. The styles are ordered from top to bottom, with the top sample produced with style 1 and bottom sample with style 12. After extensive testing, style 8 was chosen as a baseline for a human-like handwriting with patterns suitable for machine recognition. In Figure 9, style 8 was explored further with varying levels of bias in the set 0.1, 0.2, 0.4, 0.8, 1.0, 20.0. It is clear as the bias increases, handwriting legibility increases. As such, this produces a higher stability dataset for encoding and decoding data since all handwriting with bias 20.0 is nearly identical in structure in contrast to handwriting with bias 0.1.

On a further note, although not on display, samples with lower bias have higher variance in results each time the program is run. So, even though the text may be "The sky is blue" for each run, the sample with bias 0.1 would look different for the first run, second, run,



Figure 7: Sample characters from the MNIST Dataset [58, 59]

third run, and so on, whereas the samples with bias 20 would appear nearly identical to each other.

Therefore, the conclusion is made that higher biased samples will be used for this toy problem. This provides higher stability in predicting sequences of coordinates for different varieties of text inputs. However, future work includes investigating results utilizing different styles of handwriting, as well as different biases of handwriting to accommodate for the complex and wide variety of handwriting styles humans possess.

| | |
|----------|-----------------------|
| Style 1 | This is a test, Bob |
| Style 2 | This is a test, Bob |
| Style 3 | . This is a test, Bob |
| Style 4 | This is a test, Bob |
| Style 5 | This is a test, Bobr |
| Style 6 | This is a test, Bob |
| Style 7 | this is a test, Bob |
| Style 8 | This is a test, Bob |
| Style 9 | This is a test, Bobd |
| Style 10 | This is a test, Bob |
| Style 11 | This is a test, Bob |
| Style 12 | This is a test, Bob |

Figure 8: Handwriting generated in all 12 styles with Graves' work. All samples have bias = 0.8.

The sky is blue
The sky is blue
The sky is blue
The sky is blue
The sky is blue
The sky is blue

Figure 9: Handwriting generated in style 8 with Graves' work. Samples are varying levels of bias. From top to bottom, the samples have bias 0.1, 0.2, 0.4, 0.8, 1.0, and 20.0.

4.0 DATA ENCRYPTION

Encryption has existed for millenia, similar to steganography. The most basic form of encryption is a Caesar cipher [63], used famously by Julius Caesar for private communication. The basic premise is each letter in the alphabet, A-Z, is given a value 0-25 respectively. Then, each value is shifted by a specific number of letters. If this shift is a 3, all the A's in the text message become D's, M's become P's, and Y's wrap around to the beginning to become B's. Below, E_n is the formula to encrypt the message where x is the letter's value from 0-25, n is the shift amount, and $E_n(x)$ is the new character output value; $D_n(x)$ is the decryption formula [64] where the output value is the original character before the shift. As an example, the message "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG" with a shift of 3 becomes "QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD", which is obfuscated enough for any passerby without prior knowledge of the Caesar cipher. However, a trained adversary can either brute force the decoding process by trying every reverse shift until the decoded message is coherent, or they can track letter frequencies and realize 'E' occurs the most frequent in everyday corpora, so if 'B' is the most frequent letter in message with a Caesar cipher, they instantly spot the displacement amount without requiring a brute force approach.

$$E_n(x) = (x + n) \mod 26 \quad (4.1)$$

$$D_n(x) = (x - n) \mod 26 \quad (4.2)$$

Although most of encryption is founded upon mathematical analysis and methods, the growing field of neural cryptography, or making use of neural networks to learn a custom encryption algorithm, is beginning to rival against more traditional methods, such as AES [8] and

RSA [9]. The goal of encryption is to obfuscate message. Less emphasis is placed on whether the data looks normal, but rather how much required effort it takes for adversaries to crack the secret message. As such, encrypting data before steganographizing it, as proposed in this paper’s approach, supplements the already secure pipeline.

4.1 TRADITIONAL METHODS

As mentioned above, RSA and AES are the two most popular encryption schemes, representing both asymmetric and symmetric keys respectively. In security, a piece of data is typically combined with a key. This key serves as a ‘lock’ for the data and cannot be retrieved, or unlocked, unless the same key is combined with the encrypted data. Both parties, an Alice (sender) and Bob (recipient) receive the same key, but Alice encrypts the original data and Bob decrypts the encrypted data [65]. A *symmetric* key is when Alice and Bob both have possession of the same key, but they keep it private. If an adversary gains access to this private key, they instantly gain access to the encrypted data. In contrast, an *asymmetric* makes use of a private and public key for each party. If Alice wants to encrypt some data and only wants Bob to have access, she retrieves Bob’s public key from a public domain, such as their personal website or another medium. When Bob receives the message, he uses his private key to decrypt the message and retrieve the secret data. So, if an adversary discovers the encrypted data, even if they have access to both Alice and Bob’s public keys, they cannot decrypt the data unless they gain access to Bob’s private key.

As mentioned, AES (Advanced Encryption Standard) is one form of a symmetric encryption scheme. It arose out of a competition the U.S. government setup to develop a secure coding scheme to protect their classified data. Among 15 teams vying to win this competition, the Rijndael algorithm was proven to be most secure [66]. Rijndael eventually replaced DES (Data Encryption Standard), which the government used since the 1970s and realized its 56-bit key was insecure [67]. The U.S. National Institute of Standards and Technology (NIST) established AES as a standardization for all government classified information in 2001. In practice, AES is used with a 128-bit, 192-bit, or 256-bit key and is referred to AES-

128, AES-192, AES-256 respectively for shorthand. Its popularity, ease-of-use, and proof of security have led to widespread use in file transfer protocols, such as HTTPS (Hyper Text Transfer Protocol Secure), FTPS (File Transfer Protocol Secure), and SFTP (SSH File Transfer Protocol) [66].

Finally, RSA (Rivest-Shamir-Adleman) is one of the most well-known asymmetric encryption schemes [68]. Public and private keys are each formed by combining two randomly generated, unique, prime numbers. Due to the high number of bits used in the key, an adversary must factor the two prime numbers used for the private key in order to successfully decrypt the data. This method is generally secure because factoring prime numbers is class NP and several algorithms have been created to attempt to factor primes in polynomial time [69]. As computers advance in speed, keys with more bits are required to ensure the machine must exhaust all resources to factor primes. One common application area of RSA is key exchange[70].

So, AES and RSA are two common encryption schemes with two different algorithms. AES is generally faster to implement since it requires one key which is at most 256 bits. However, RSA requires less effort on both ends since they do not require the same private key. Despite these differences, both are in global use.

4.2 NEURAL CRYPTOGRAPHY

Neural cryptography, to match the growing practicality of neural networks, is the practice of encrypting data with the use of machine learning models. [71, 72], in 2002 and 2013, are two of the first introductions to the field. They both propose and analyze a tree parity machine with a binary output. Assume three constants exist: K determines the size of the tree parity machine, N is the size of the input data in bits, M is the input message to the encrypting network, and M' is the output message of the encrypting network. A tree parity machine receives $K * N$ neurons as input with values in $\{-1, 0, +1\}$ (Figure 10 has $K = 3$), passes them through a typical multi-layer feed-forward neural network, and outputs a binary response in $\{-1, +1\}$, as seen in Figure 10. This requires no public key from either the

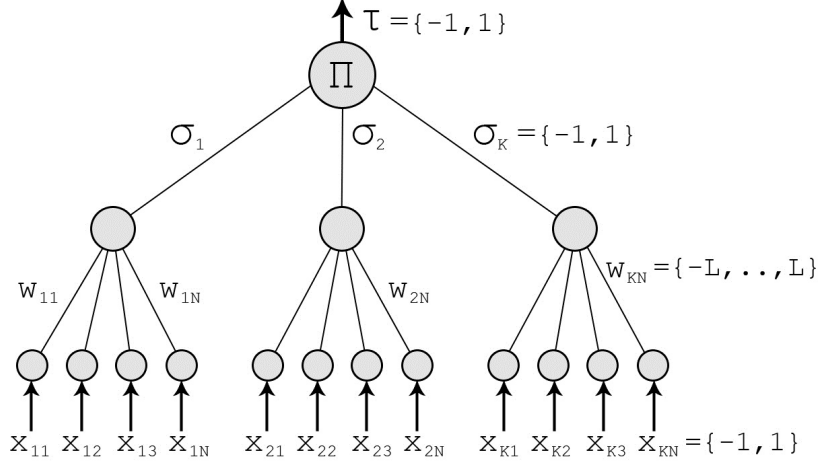


Figure 10: Tree Parity Machine

encrypting or decrypting party, but requires the decrypting party to train a neural network by passing input M into the encrypting network, reading the output M' , inserting M' into their network, and minimizing a loss function to output the same, original input value M . [73] improves on this approach with a permutation parity machine, which works on binary, rather than trinary inputs. Meanwhile, [74] proposes genetic, geometric, and probabilistic attacks for cryptanalysis. Possible shortcomings with above approaches is they condense, rather than propose a new form of the data, in order to encrypt data. Additionally, they are not trained against adversaries, but are explored in terms of a recipient training its network for key exchange with the sender's network.

More recently, [75] proposes the network in Figure 4.2. The typical Alice-Bob communication, with Eve as an eavesdropper, is replaced by a series of three multi-layer feed-forward neural networks. These neural networks are based on the the premise of autoencoders [76, 77]. Autoencoders are generally used for denoising images [78] and dimensionality reduction [79] by reducing dimensionality of input I of size N to a latent space representation of size M where $M < N$. Then, given M , the I can be fully reconstructed. In the case of neural cryptography, the input representation combines both a key K of S bits and data D of T bits into a feature vector of size $S + T$. This is passed through the feed-forward network to

output a latent space representation D' of size T , which is the same as the original data. This first network can be viewed as an ‘encrypting’ network. With D' , the K and D can be recovered with a similar ‘decrypting’ network. A simplified version of this process can be seen in Figure 11.

Although success is easily achieved with Alice and Bob, the learned representations are not robust to adversaries since Alice can simply force the latent space representation to appear identical to the original data D , allowing Bob and any adversary to recover the data easily. As such, a third network, Eve, is added to the mix. A key and data are given to Alice to produce some output representation. Then, Bob receives the encrypted data and the key and learns how to decrypt the message. As an eavesdropper, Eve only receives the encrypted data and attempts to recover the secret information. [80] proposes a slight twist to this scheme. Instead of Eve only attempting to decrypt data, she also receives two pieces of data (P_0 and P_1), one of which has been encrypted and the other untouched, and outputs 0 or 1 depending on which output she believes has been encrypted. If P is the original data, K is the key, C is the ciphertext, or encrypted data, $Eve(C)$ is Eve’s decrypted data, $Bob(C, K)$ is Bob’s decrypted data, and $d(P, P') = \|P - P'\|_1$, then the losses are as follows:

$$L_{Eve} = d(P, Eve(C)) \quad (4.3)$$

$$L_{Bob} = d(P, Bob(C, K)) \quad (4.4)$$

$$L_{Alice} = L_{Bob} - L_{Eve} \quad (4.5)$$

where L_{Bob} can be seen as the decryption, or reconstruction error, L_{Eve} can be seen as the adversarial reconstruction error, and L_{Alice} can be seen as the maximization of the distance of these errors. The goal is to minimize Bob’s reconstruction error and maximize Eve’s reconstruction error. When encrypting sixteen bits of data, Bob’s error is zero bits, while Eve’s error is eight bits (50%), indicating Eve does no better than random guessing. The advantage to this scheme is that the encrypted data is forced to output discrete, integers in $\{0, 1\}$ rather than continuous values in $[0, 1]$. So, there is less suspicion of the data being

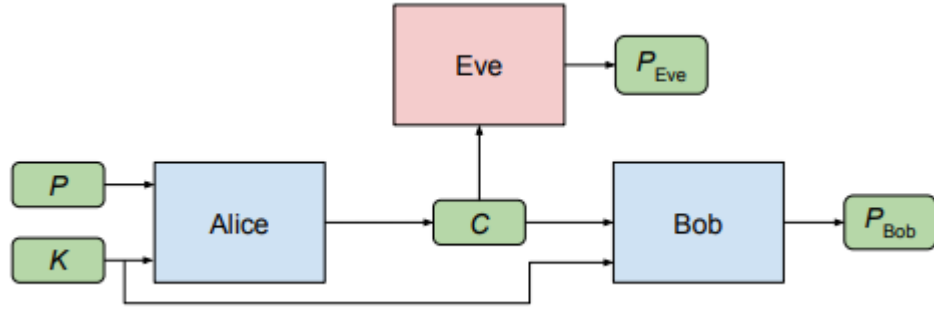


Figure 11: The neural cryptography network proposed by [75]

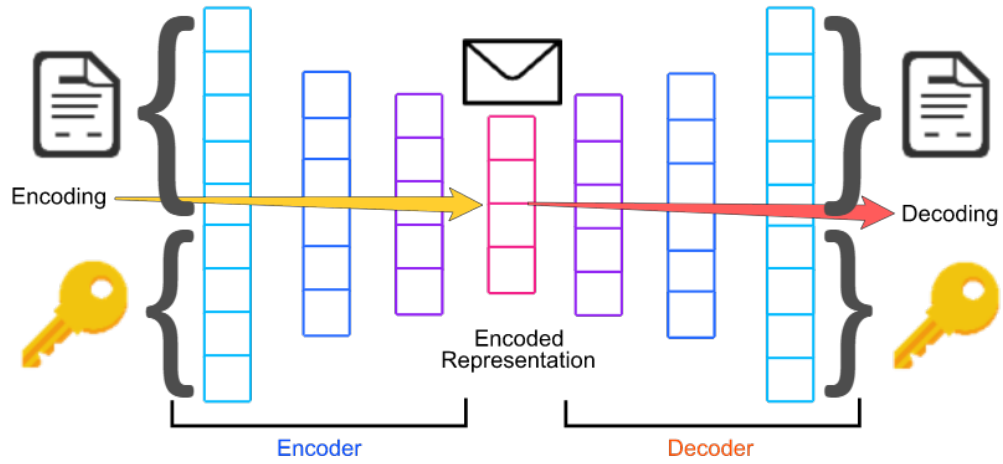


Figure 12: High-level overview of neural cryptography autoencoder

encrypted if it is simply in the form of discrete bit values.

4.3 DATASET

The dataset for the neural cryptography task, as mentioned above, is simple. In general, the data can be infinitely generated as n bits (0 or 1) with an n bit key. However, for each training batch, data is generated as integers in $\{-1, +1\}$ instead of $\{0, 1\}$. This allows for the use of the tanh activation function in training the neural networks, rather than a sigmoid activation. The advantage of using tanh is having higher gradients for backpropagation [81]. As such, if $b \in \{-1, +1\}$, input feature vectors are generally seen as follows:

$$P = \langle b_1, b_2, \dots, b_{n-1}, b_n \rangle \quad (4.6)$$

In this paper, n is fixed at 128 bits for both the data and key, forming an input vector of size 256 bits:

$$P = \langle b_1, b_2, \dots, b_{256} \rangle \quad (4.7)$$

4.4 METHODS

The approach in this paper is similar to the above neural cryptography methods. However, instead of training three networks (Alice, Bob, and Eve), four networks are trained (Alice, Bob, Eve, and Dave). The new network, Dave, serves as an additional adversary. One issue with the aforementioned neural cryptography approaches is Alice and Bob are trained to be robust against Eve, but there is no proof of security against general adversaries. As such, while Alice and Bob are training for robustness against Eve, Dave loss is independent of Alice and Bob's, so he is an independent network attempting to decrypt the data. This new network of agents can be seen in Figure 13.

In this sense, this network performs encryption, decryption, and cryptanalysis. The individual networks of the individual agents differ slightly. Alice receives both the original

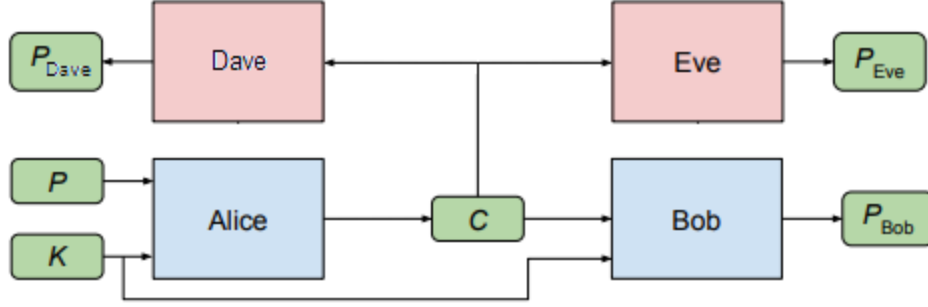


Figure 13: The new, proposed neural cryptography network with Alice, Bob, Eve, and Dave

data and key, which are passed into a network with one hidden layer of $256 \times 192 \times 128$. Bob is built with the same architecture. He receives the encrypted data and key, resulting in the following network: $256 \times 192 \times 128$. Eve only receives the encrypted data and is built with three layers (input, output, and one hidden layer): $128 \times 192 \times 128$. Finally, Dave is built with an identical structure to Eve. No convolutional layers were used in the network architecture since the data is randomly generated, so it lacks the spatial structure convolutional layers attempt to extract, for example in convolutional neural networks. The entire network was trained with a learning rate of $\alpha = 10^{-6}$ for 10,000 epochs of 2,000 iterations per epoch with stochastic gradient descent using the *GradientDescentOptimizer* in TensorFlow [82], resulting in 20 million step iterations. Both the *Adam* and *RMSProp* optimizers were tested, but results were either equivalent or worse than stochastic gradient descent with *GradientDescentOptimizer*.

The goal of Eve and Dave is to ensure their reconstruction errors are 0 bits on a scale from 0 to 128 (he successfully retrieves the entire 128-bit original message), or 0%. Meanwhile, Alice and Bob want Bob's reconstruction error to be 0 bits (0%) while ensuring Eve and Dave's reconstruction error is 50%, or 64 bits. If Eve and Dave had a 100% reconstruction error, they could simply flip their predicted bits to have a 0% reconstruction error. So, an error of 50% is equivalent to Eve and Dave random guessing. The losses of the agents are below, where L_{Eve} , L_{Dave} , and $L_{Bob/Alice}$ are the four respective agents' losses, P_{Orig} is the original data, P_{Bob} , P_{Eve} , and P_{Dave} are the decrypted data from the three agents, and P_i is

the i^{th} bit of the respective piece of original or decrypted data:

$$L_{Eve}(P_{Orig}, P_{Eve}) = \sum_{i=1}^{128} |P_{i,Orig} - P_{i,Eve}| \quad (4.8)$$

$$L_{Dave}(P_{Orig}, P_{Dave}) = \sum_{i=1}^{128} |P_{i,Orig} - P_{i,Dave}| \quad (4.9)$$

$$L_{Bob/Alice}(P_{Orig}, P_{Bob}, P_{Eve}) = \sum_{i=1}^{128} |P_{i,Orig} - P_{i,Bob}| + \left| 64 - \sum_{i=1}^{128} |P_{i,Orig} - P_{i,Eve}| \right| \quad (4.10)$$

It is important to note the neural networks use a tanh activation function for each layer, so the output neurons, or predicted bits, are continuous values. So, the losses are continuous values, but final predictions of each network in a practical setting require rounding the continuous values to the nearest integer in $\{-1, +1\}$, adding 1 to the bit value, and dividing by 2 to transfer the domain to $\{0, 1\}$. All loss functions are minimized.

Bob and Alice's loss is important. The first term is Bob's reconstruction error, which is the non-negative L_1 error between the original data and the decrypted data. The second term is the absolute distance between 64 and Eve's non-negative L_1 distance from the original message and her decrypted message. In other words, if the loss is minimized, the sum is minimized, which means the two terms are minimized. In order to minimize the first term, the Bob's reconstruction error will be 0 bits. In order to minimize the second term, Eve's reconstruction error will be 64 bits (on a scale from 0 to 128), resulting in a 50% reconstruction error. If the terms were subtracted, there is incentive for the second term to be maximum distance from 64 bits, resulting in Eve having 100% reconstruction error or 0% reconstruction error and a loss value of -64. Or, it might result in Eve's reconstruction error being 64 bits while Bob's reconstruction error is 64 bits, resulting in a similar loss value of -64. Neither result is desirable, so the loss function was precisely tuned for the necessary above conditions. Finally, the benefits of using L_1 norm between the original message and the decrypted message, in contrast with mean squared error (MSE), mean absolute error (MAE), or L_2 distance, is marginal but useful in interpreting errors. If an error of 1.2 bits is achieved with L_1 error across 128 bits, no bit is off by more than a value of 1.2. Meanwhile,

if the average of the offsets was calculated and it came out to 0.4, most of the bits might have a small distance, but a few bits might be completely off. As such, with MAE or MSE, error rates smaller than 10^{-3} must be achieved to be confident in results.

4.5 RESULTS

Impressive results were achieved with the three agents showing robustness against two adversaries. Moreso, considering Bob was not trained to be robust against Dave, the results are even more notable. Figure 14 showcases the learning curve of Alice, Bob, Eve, and Dave. Table 1 displays the actual loss values of the agents after training finished.

As observed, Alice and Bob achieve their optimal state. Their combined loss converges to 0%, indicating Bob’s reconstruction error is close to 0% and Eve’s reconstruction error is close to 50%, which is reflected in Eve’s loss plot. Additionally, Dave’s loss converges to almost exactly 50% which showcases the robustness of Alice’s algorithm. The fact an adversary without access to the ground-truth data or key is unable to recover the secret data exhibits Alice and Bob are able to learn a secure, 128-bit cryptographic scheme while only training against a single adversary. Therefore, even if an adversary had access to the outputs from Alice, they are unable to successfully retrieve the original data unless they have access to a key.

With that being said, these agents utilized somewhat trivial networks with two fully-connected layers. It is suspected if Eve and Dave were deep neural networks [83], they may stand a better chance at cracking the code. However, Alice and Bob may be able to counter this by extending their architectures to learn more advanced schemes. Further testing may be appreciated but is not the aim of this paper, so exploration was stopped with the above satisfactory results.

The input to the neural network is a 256-dimensional feature vector, half of which contains the binary data and the other half containing the binary key. As such, it can encode any form of data as long as it can be transformed into a binary representation. For a given data representation, there are 2^{128} combinations in which a piece of data can be scrambled.

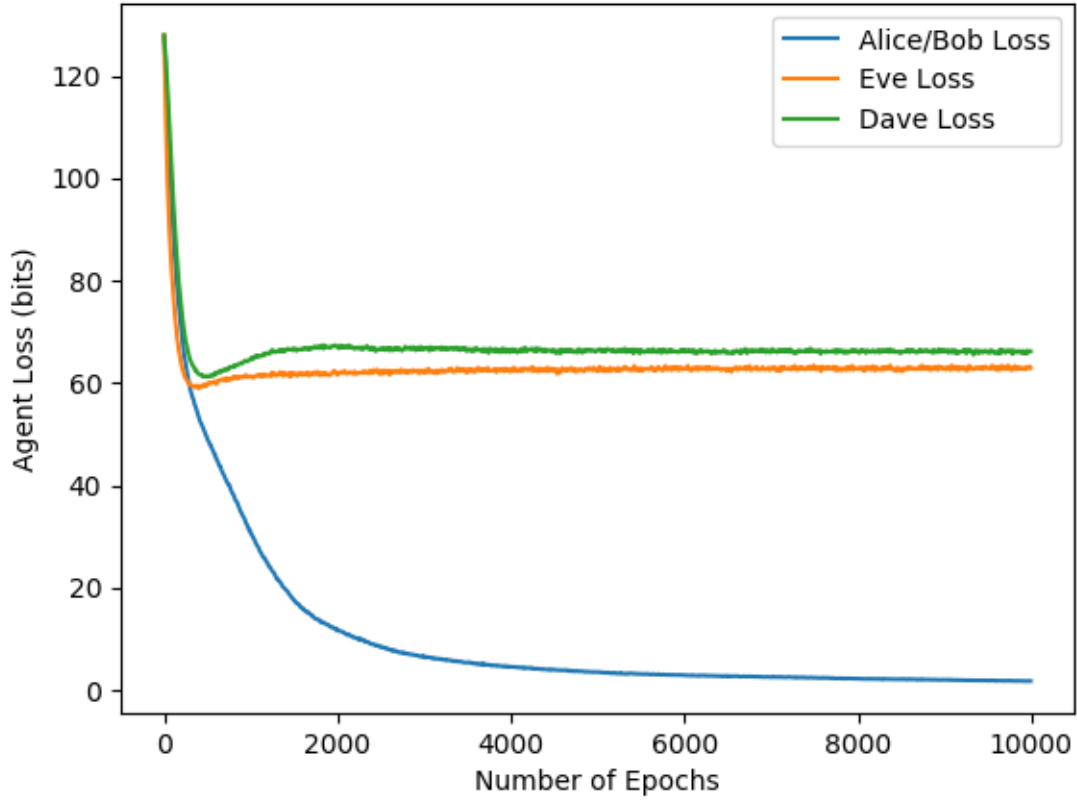


Figure 14: Loss of the Alice/Bob, Eve, and Dave agents. Loss is recorded in bits and is on a scale from 0 to 128, where 0 represents perfect reconstruction error and 128 represents the worst reconstruction error. Bob/Alice aim for a loss of 0 while ensuring Eve has a loss of 64 bits, equivalent to random guessing. Eve and Dave attempt to achieve a loss of 0 bits.

This neural network scheme is able to learn those 2^{128} representations within a matter of hours. This reduction in time is possible because a neural network utilizes gradient descent on a smaller sample size rather than memorizing all 2^{128} possible combinations. Therefore, it efficiently learns how to encrypt high-dimensional data with a fraction of the samples. It is not feasible to brute force this network due to the exponential number of combinations for a given input data. Future experiments might be worthwhile to investigate using a 75/25

| | Alice/Bob | Eve | Dave |
|-------------|-----------|----------|----------|
| Loss (bits) | 1.7569 | 62.8310 | 65.5122 |
| Loss (%) | 1.3726% | 49.0867% | 51.1814% |

Table 1: Precise loss values after training all four agents for 10,000 epochs of 2,000 iterations per epoch. Additionally, these loss values are divided by 128 as a form of normalization to determine the percentage of bits each agent successfully recovered.

data/key split instead of a 50/50 split (128-bit data and 128-bit key) in terms of adversaries learning the coding scheme.

When the data is passed into the neural network, 128 continuous-valued neurons are output. Although not tested in this framework, an additional adversary to determine the difference between an encrypted and non-encrypted message would be useful, similar to [80]. This might encourage Alice to output binary values instead of continuous values since Eve can spot any continuous value and assume the data has been encrypted. However, this avenue was not pursued because if an adversary can detect this paper’s approach is steganographic in nature, the natural conclusion is the data is encrypted as well.

As a result, the neural cryptography aspect of this pipeline is a success since Alice and Bob can learn a symmetric encryption scheme. The encrypted, continuous outputs can then be combined with existing handwriting coordinates to provide new offsets. The continuous characteristic is not a concern, Alice and Bob learned an efficient coding scheme, the coding scheme was robust to a present adversary, and the coding scheme was robust against an arbitrary adversary.

5.0 STEGANOGRAPHY

Steganography has existed for millenia and has strong historical roots in general message passing, message transmission during wars, and digital media message hiding. The field deals with the concealment of hidden information, while also ensuring adversaries do not suspect the existence of hidden information. Often seen in other fields of security and computer science, common applications use images and much of the recent research is in the field of computer vision. Generally, an image A is referred to as a cover image, message M is referred to as the secret image/message, and output image B, the result of combining A and M, is referred to as the container image. Although often utilized to hide, or embed, information on a pixel-by-pixel basis in traditional algorithms, as discussed earlier, this paper utilizes steganography to modify the contents of the image, namely the actual location of handwriting stroke coordinates.

5.1 HISTORICAL BACKGROUND OF STEGANOGRAPHY

To understand the significance of steganography, the natural introduction is to analyze its neighbor, cryptography. Cryptography is the study of algorithms and codes capable of *obscuring* information such that the codes cannot be broken by an adversary. A common applied form of cryptography is encryption, as discussed in the previous chapter [84, 85]. Meanwhile, steganography as a field formed as early as 440 B.C. when Histiaeus sculpted a message into his servant's head and ordered him to deliver it to a vassal [2]. The term was formalized by Johannes Trithemius, a German polymath, in his 1499 book Steganographia, which was a steganography book fittingly disguised as a book on magic [86]. As such, it is

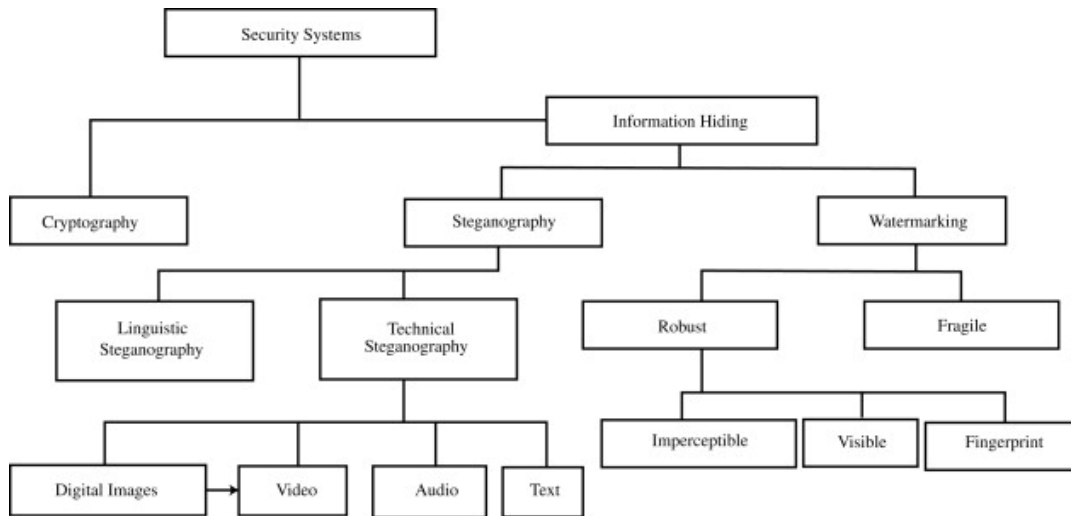


Figure 15: Breakdown of data security and concealment fields [87]

based on hiding the fact that secret data exists, rather than obfuscating it.

Steganography and cryptography are two similar fields concerned with concealing hidden information, as seen in Figure 15, but have one significant difference. Cryptography is concerned with transforming a hidden message into indecipherable jumbled output data. When a passerby encounters this jumbled data, they can generally notice, visually or analytically, that the data is not normal. They determine the data has been transformed previously. In this sense, cryptography is optimizing for one task, which is preventing adversaries from breaking, or reverse engineering, a secret code or formula which transformed some input data. If the adversary cracks this secret formula, they easily gain access to the original data. Meanwhile, steganography is concerned with flipping the script. The primary goal of steganography is to combine secret data with normal data. If successful, the combined, output data resembles the original, normal data as closely as possible in the hopes that an adversary looks at the data and convinces themselves the data has not been modified. Therefore, if the data passes this initial human inspection, the risk of an adversary even attempting to break the code or formula is avoided and the secret data is secure. Both cryptography and steganography have different end goals, with cryptography generally being the easier

task because the formula for transforming the secret just needs to be extremely complex. In technical terms, cryptography needs to minimize decryption error and maximize adversary error. Meanwhile, steganography needs to minimize the decryption error, maximize adversary error, and minimize the container error. In one sentence, cryptography is concerned with only protecting the message contents, while steganography is concerned with concealing the message contents and hiding any indication a secret message exists. Namely, steganography aims for success in two tasks:

1. **Human visual inspection:** can a human examine the image and notice another image, or some data, is embedded? If not, the steganography algorithm is successful.
2. **Machine inspection:** can a machine use popular steganalysis tools to identify patterns in the container image to decode the secret message? If not, the steganography algorithm is successful.

The practice has been used for a variety of purposes. For example, LSB is commonly used for digital watermarking [88, 89, 90, 91]. However, these watermarking procedures are generally designed to be easily seen. An advanced form of watermarking is using steganography to embed the author’s signature or data. Then, in copyright cases, the author can extract the embedded information to prove their case [92]. Another use for this secretive watermarking is in the film industry. For example, if a film studio pre-leases a movie to its employees, they want to prevent leaks. As such, if the studio stamps the film with a unique watermark for each employee, when the film is leaked, it can be traced back to the perpetrator. Finally, using steganography, a hospital or medical facility can embed the patient’s medical record or medical history into X-rays to ensure secure transmission and storage of their data.

With the invention of modern computation in the 20th century, steganography immediately transformed from old, traditional, manual practices with elaborate schemes to direct embedding into information. This came in the form of the least-significant-bit steganography algorithm, which is commonly referred to LSB steganography. The approach is rather naïve and easily broken, but is a fast, efficient, and simple method to carry out the process. Figure 16 showcases the general idea of the process. This process has been discussed previously, so

its technical details will not be discussed in this section.



Figure 16: The process of least significant bit (LSB) steganography [93]

Figure 17 displays 2-LSB steganography with an image of a cat with the secret message "The password is 123". There are two avenues to decode this message. The first approach is to communicate between two adversaries whether 1-LSB, 2-LSB, or 3-LSB was used; this is more likely for communication between a sender and recipient. The second approach is to simply scan the container image from top left to bottom right, reading the red, then green, then blue channel of each pixel, sequentially extracting the two least significant bits by simply ANDing the 8-bit channel value with 00000011; this is more likely for a steganalysis tool or adversary. Then, after extracting those bits, combine the 2-bit chunks into 8-bit chunks to form an 8-bit ASCII character to retrieve the message "The password is 123". This message is 19 characters long, which equates to 152 bits of information. If using 2-LSB steganography, it would take $\text{ceil}[(152 \text{ bits}) / (2 \text{ bits per channel}) / (3 \text{ channels per pixel})] = 26$ pixels to hide the information. It is important to note the value n in n -LSB steganography dictates how many of the 8 bits to replace with the bits in the secret

message. Fittingly, if $n = 1$, a cover pixel’s value will change by a maximum value of 1, ensuring the output container image is as close to the original cover as possible. However, if $n = 8$, all 8 bits will be replaced with the information from the secret message, resulting in a nearly completely randomized static image. In practice, it is common to see $n = 2$, which results in a 25% embedding ratio. In other words, 25% of the image is replaced by a secret message. Although simple and fast, that dictates its own downfall. Steganalysis tools commonly break through this simple algorithm by scanning 1-LSB, 2-LSB, and 3-LSB to find any hidden information. In fact, most modern steganalysis tools check LSB embedding as a first form of decoding secret data in steganographic images. [94] is an open-source repository on GitHub for specifically detecting LSB steganography. Meanwhile, [95] and [96] are other privately-owned or university-owned conglomerates of steganalysis tools, containing a few with functionality for decoding LSB steganography. Therefore, while it works, it is rarely used in practice due to its immediate shortfalls.

5.2 NEURAL STEGANOGRAPHY

Neural steganography, as discussed in the related work of this paper, has already been slightly detailed with the introduction of [1]. Besides [1], there is one more notable work worth mentioning. [17] utilizes machine learning approaches to also embed information in the image. They utilize convolutional autoencoders [97] to compress information and its spatial structure into a cover image. Then, the spatial structure of the secret image is recovered with the decoder. Their approach is robust to noise and small perturbations in data, producing data recovery rates of 10^{-5} . Their steganographic image, when compared to a non-steganographic image, was detected of being steganographic in nature about 50% of the time, beating other methods with 68% and 78% detection rates. However, there is some suspicion with further testing this detection rate might not be as superior to other methods if given a ‘choose 1 of 3’ scenarios. For example, if a network was shown 3 images and told to predict which was steganographic, a lower accuracy is best for the steganographic team. However, [17] aims to achieve a 50% accuracy due to the binary nature of the problem. It is



Figure 17: Left: original cat image. Right: steganographed cat image using 2-LSB steganography on display at [93]

under high suspicions modifying the structure of the data will result in a lower steganalysis accuracy than [17]. Additionally, their work produces a capacity, or embedding ratio, of 0.203 bpp, similar to the approach discussed in the following sections. As such, they mostly differ in the nature of how the data is modifying the image, while having the same, correct idea in mind of focusing on data recovery rather than embedding ratio as shown in [1].

In terms of steganalysis, simple methods such as StegExpose were previously mentioned as limited in nature since they test against pre-defined steganographic algorithms. When neural steganalysis tools are utilized, [1] fails with a 90-99% detection rate due to its high embedding ratio. Meanwhile, [17] utilizes a slightly different approach, ATS, which generates training samples by producing container images with the steganography network, training

a supervised, binary classifier, and predicting whether an image is steganographic or not. In their case, a single binary classifier trained on the underlying dataset produces a 98% detection rate, which is a weakness. To curb this, they explore utilizing five different classifiers trained with separate weight initializations. The ATS classifier produces its training set images through these five classifiers. A sixth classifier is trained with a unique initialization and ATS eventually produces a 50% detection rate with this model’s output images, showing its robustness as long as an adversary does not possess the steganography network’s weights.

5.3 IMAGE VS. NON-IMAGE DATA

With the proposed steganographic approach, 8-bit ASCII characters are “steganographed” into the image. Naturally, there are unique differences between image and non-image data. First, when hiding a secret image into a cover, they both have spatial structure. This makes the use of convolutional neural networks more applicable since they inherently preserve that spatial structure. Second, if the secret data is in image form, the extracted image, even from the correct recipient, will contain several pixels with large differences in intensity from the original secret image. In fact, [1]’s output had an average pixel intensity difference of 3 values. Although they embedded a large amount of information, if the data was text rather than pixel intensities, having an error of 3 ASCII values might turn an ‘a’ into a ‘d’ or ‘M’ into ‘p’, essentially scrambling all the data, nullifying the effect the approach. As such, text data, and especially ASCII, which is used in this paper, requires more care in order to extract the correct data.

5.4 METHODS

It is important to remember the goal of the steganography portion of this paper is to *modify* the contents/metadata/characteristics of the image to embed information, rather than directly embed information as in typical steganographic approaches. In order to accomplish

this task, the coordinates of handwriting are slightly modified with some offset depending on the contents of the secret data combined with it.

The method is simple. An 8-dimensional feature vector is formed from an original x and y coordinate, 2-bit data vector, and 4 biases. There is one fully connected layer to the output layer, which is two-dimensional and represents the new coordinate (x', y') . To recover the 2-bit data vector, another network is built for extracting the data from the transformed coordinate. A 6-dimensional feature vector is formed from x' , y' , and four bias terms. This input layer is fully connected to a hidden layer with 8 neurons, which is fully connected to a 2-dimensional output layer with two neurons representing the two-bit data embedded in the coordinate. Any coordinates in question are real-valued 64-bit floating point numbers.

The two networks were trained in tandem for 700 epochs of 2,000 iterations per epoch, totalling 1,400,000 total samples. Input coordinates were 32-bit integers converted to 64-bit floating point numbers in the discrete interval $[0, 2000]$, representing most reasonable images generated for handwriting. The two-dimensional data vector concatenated with the input coordinates are two randomly generated bits in $\{0, 1\}$. A learning rate of 10^{-6} was utilized using stochastic gradient descent and the *GradientDescentOptimizer* in TensorFlow [82]. Both the *Adam* and *RMSProp* optimizers were tested, but results were either equivalent or worse than stochastic gradient descent with *GradientDescentOptimizer*.

Finally, in order to optimize and secure the data in two coordinates such that the output coordinates are as close as possible to the original coordinates, two different losses were used. The first, for the encoding network, is a simple L_1 loss for the x coordinate and for the y coordinate; this is the cover loss. Then, for the decoding network, the loss is the L_1 distance between the original data and the secret data; this is the decrypt loss. Losses can be seen below, where x and y are the coordinates and b is the respective bit values of the data:

$$L_{cover}(x, y, x', y') = |x - x'| + |y - y'| \quad (5.1)$$

$$L_{decrypt}(b_1, b_2) = |b_1 - b'_1| + |b_2 - b'_2| \quad (5.2)$$

$$L_{total}(L_{cover}, L_{decrypt}) = 2 \times L_{decrypt} + L_{cover} \quad (5.3)$$

In practice, a bit is a discrete value in $\{0, 1\}$, but a neural network outputs a continuous value, so the decrypt loss makes sense. Also, even though coordinates are discrete values (cannot have $\frac{1}{3}$ of a pixel), graphics editors take care of this by adding neighboring grey pixel values. So, if the coordinate in a handwritten sample is $(134.0, 26.0)$, there would be a black dot at that specific coordinate and white surrounding it. However, the coordinate $(134.5, 26.4)$ has a mix of light and dark grey colors in a 2×2 area rather than one definitive black pixel, giving the illusion the pixel's continuous location is between the four discrete values. This continuous property is unable to be replicated in traditional steganographic approaches since pixel intensity values are discrete from 0 to 255.

5.5 RESULTS

The resultant losses can be seen in Figure 18 and Table 2. The hope is the cover error, or the deviation of the new (x', y') coordinate pair from the original (x, y) is small while also being able to successfully recover the 2 bits of data encoded in the coordinate. The decoding loss is the deviation of the extracted data from the true, original values measured with the L_1 norm.

One will notice the decoding error is near 0 with an average error of 2.9×10^{-3} . It is important to note the 2 bits of data passed into the steganography network, along with the coordinate, are a small chunk of the larger encrypted data discussed in earlier chapters. As such, if the decoded values were binary in $\{0, 1\}$, rounding would solve any issues, even if the decoding error was as high as 0.2. However, due to the nature of the neural cryptography network, a deviation on the scale of 10^{-2} or 10^{-3} across 128 bits may cause significant issues when decrypting data. As such, when weighing these two terms in the total loss function, decoding error has higher priority. Moreso, we do not want the cover error to be exactly zero since that makes detecting steganographized coordinates more difficult later on in the pipeline for the decoding direction. As such, having $0.1 \leq L_{total} \leq 0.3$ ensures some deviation without being suspicious. We can see the cover loss is 0.1770 bits, indicating the L_1 distance between the new coordinates and the original coordinates is 0.1770 on average. Meanwhile,

the decoding loss is 0.0029, indicating the L_1 distance between the decoded bits and the original, encrypted bits fed into the network is 0.0029 on average. The reason this is not zero is because a neural network outputs continuous values, so instead of directly outputting the originally fed in data of $\{1, 0\}$, the steganography network might output $\{0.9992, 0.0021\}$, which are close approximations.

One issue with this steganography network is the low embedding ratio. Only two bits were able to be embedded successfully while maintaining a low decoding and relatively low cover error. While this is similar to the widespread, practical application of LSB, there are still some advantages. For example, LSB’s embedding capacity is limited by $N \times N \times 2$ bits assuming 2-LSB. But, coordinates can be artificially represented as continuous values in pixels with the help of grey shading and pixels. So, technically, if a coordinate was placed on every other pixel of an image, the coordinate can be embedded in 4 different fashions, resulting in $N \times N \times 2$ bits of information in the image. In practice, this will never happen, but hypothetical limits can be pushed. In addition, LSB is stable in the sense any retrieved data or bits are the ground-truth values, unlike current neural steganography approaches. However, LSB is easy to detect, crack, and retrieve data. As such, displacing coordinates is able to retrieve the ground-truth data values while also achieving a more cunning result in terms of steganography by displacing the coordinate, therefore modifying the image rather than embedding information. Finally, despite these concerns, [17] achieves a similar bpp (bits-per-pixel) ratio of 0.203. Bits-per-pixel is measured as $L/(C \times H \times W)$ where L is the number of bits of the message, C is the number of color channels in the image, and H and W are the height and width, respectively, of the image while achieving error rates close to 10^{-5} , indicating high recovery of precious data.

This approach is also secure since the network learns its own displacement values for

| | Cover | Decoding | Total |
|-------------|--------|----------|--------|
| Loss (bits) | 0.1770 | 0.0029 | 0.1800 |

Table 2: Precise loss values after training all for 690 epochs of 2,000 iterations per epoch.

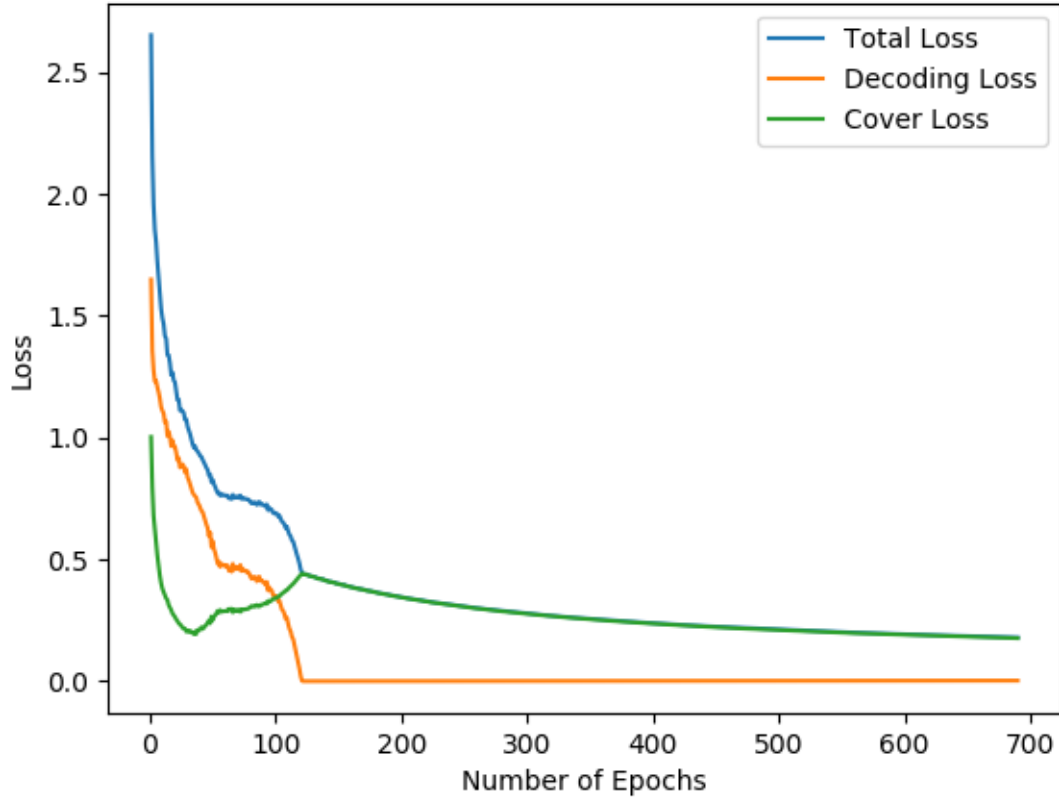


Figure 18: Values across all 690 epochs for the cover, data decoding, and combined total loss.

each 2-bit data combination on each (x, y) coordinate. So, even if an adversary detects the coordinates in the image, they must know the weights to the network in order to desteganographize the coordinate. The task becomes even more difficult even if an adversary manages to extract coordinates from the image. As such, this is a secure approach that requires an adversary to possess the network's weights in order to decrypt the message.

6.0 ENCODING & DECODING DATA

The purpose of discussing Chapters 3, 4, and 5 is to construct the foundations of the steganographic pipeline. Not only do the individual parts (i.e., encryption, handwriting generation, and steganographization) have to work, but integration is key as well. For example, if the neural networks for encryption output continuous values, the coordinate steganographization must handle continuous values. Otherwise, the processes must be restructured to fit the input and output specifications. In this section, the pipeline and integration for encoding and decoding is discussed to ensure practicality of implementation of this method to solve the toy problem.

6.1 PIPELINE & INTEGRATION

First, encoding data is discussed. This forward pass is relatively straightforward and simple. The general process is as follows:

1. Encrypt data M to produce M'
 - a. Covert data to binary
 - b. Split data into chunks S_i of 128 bits
 - c. Generate random 128-bit key K
 - d. Encrypt each S_i with K to produce M'_i
 - e. Concatenate all M'_i to produce M'
2. Generate coordinates C of realistic handwriting
 - a. Choose arbitrary cover message (e.g., “The sky is blue”)

- b. Input cover message into RNN to produce coordinates for cover image
- 3. Produce new coordinates C'
 - a. Split M' into chunks of 2 bits
 - b. Sequentially combine C and each 2-bit chunk to produce C'
- 4. Plot C' on canvas to output new handwriting container image

The process is a rather straightforward encoding pipeline for this steganographic approach. Many aspects in terms of integration have been discussed, such as the continuous neural cryptographic outputs being fed into the steganographic network. As such, this outline is left by itself with no further explanation. Now, once encoding is finished, the difficult process is decoding the data. This involves recognizing the beginning of a sequence, and predicting future coordinates which have been displaced from their traditional values. As such, the reverse task is magnitudes more difficult than encoding.

It is important to mention the threat model at this step. A threat model outlines the assumptions of attackers and adversaries. In this work, the threat model is quite simple. All adversaries have access to the network's outputs. For example, an adversary may pass secret data into the pipeline along with a cover image to create a container image, but they do not have access to the underlying weights of the three neural networks (RNN for handwriting generation, neural cryptography network, and coordinate steganography network). As such, the intermediate step in the process, encrypting data (before it is hidden in the coordinate displacements) is not output to the user. This creates two issues for an attacker. First, since the weight initializations the neural networks differ by a random seed, an adversary is not guaranteed to converge to the same local loss minimum when performing gradient descent. As such, this may require them to endlessly re-train the networks. Second, even if the coordinates are detected from the container image, assuming an adversary successfully learns the weights of the steganography network, they must successfully decrypt continuous values output by the encryption network, into discrete, binary bit values, which is not a trivial task. Additionally, all these attacks assume the adversary realizes secret data is present, or in other words, the container image is steganographic. The non-guarantee of converging to the same local minimum of weights through gradient descent and possible use of cipher block chaining, compared to the current electronic codebook encryption scheme, both mentioned in

the future work, ensures no repetition or patterns in encrypted data. Additionally, the single task of recovering the coordinate locations is not trivial as every coordinate is a continuous value in the image, not discrete, so the handwriting looks continuous, thus resulting in a near infinite combination of coordinate sequences to model the handwriting. So, even if the threat model assumes an adversary has knowledge of the neural network architectures utilized in each step of the steganography pipeline, the current steps have several defenses against attacks, resulting in a robust pipeline.

6.2 RECOGNIZING & DECODING THE SEQUENCE

The decoding process appears to be a bit more straightforward than the encoding process in Figure 22, but in fact requires more thought. First, the image must be read into memory, then the initial steganographed coordinate must be recognized, and using the knowledge of previous coordinates, in addition to a shifting window of image features extracted with the use of a CNN, future coordinates can be recognized. To round out the process, the encrypted data is extracted from the steganographed coordinates, and the data is decrypted to retrieve the original data. The process is as follows:

First an image is read into memory, such as Figure 19. This particular image is generated with 472 coordinates, which is an average of 39 data points for each of the 12 letters in “The sky is blue”. It is important to note this image has dimensions 510×100 . If a different message were used for the cover image, such as “My dog is tall and is very playful” (27 letters), the image dimensions would be roughly twice as wide with the same height. This can be extended for as many resources the given machine possesses, but can easily be split into several generated lines, such as shown in Figure 20, which uses 1453 data points and produces a 610×230 image.

The next step is discovering the beginning point of the coordinate sequence. It is important the coordinates are recognized in their respective order of generation since that is how the encrypted data is laid out. This process is completed by the use of a convolutional neural network. Specifically, the architecture is similar to LeNet [98] (Figure 21), a foundational

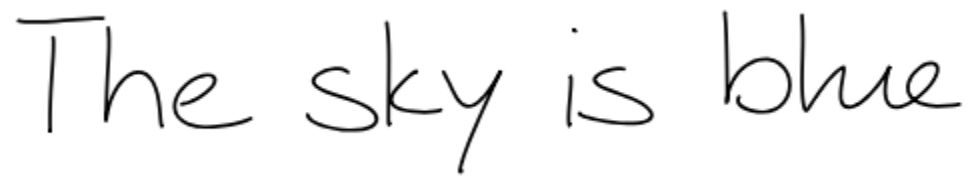
A sample of human-like handwriting in black ink on a white background. The text "The sky is blue" is written in a cursive, slightly slanted style. The letters are connected, and the overall appearance is that of a casual, handwritten note.

Figure 19: An original generated sample of human-like handwriting before steganographization. This can be viewed as the cover image in steganography jargon.

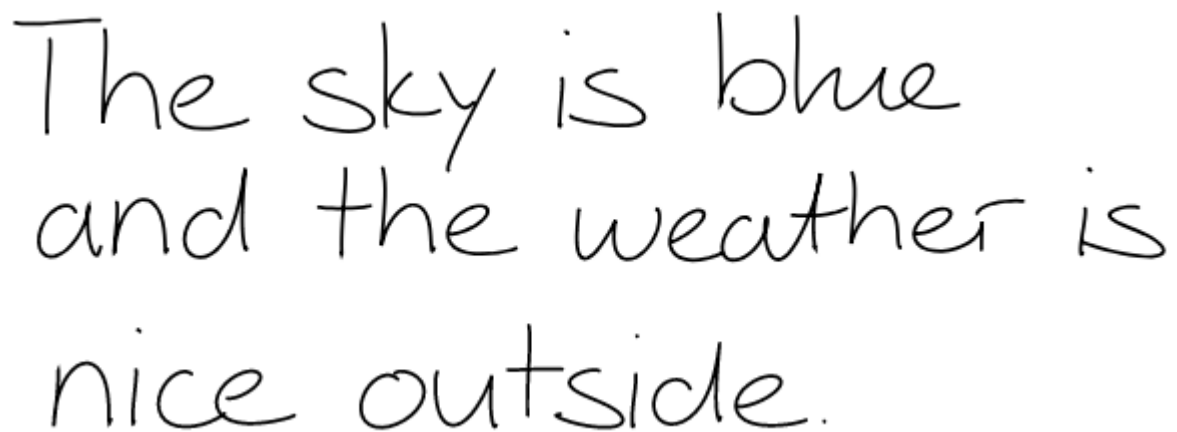
A sample of human-like handwriting in black ink on a white background. The text "The sky is blue and the weather is nice outside." is written in a cursive, slightly slanted style. The letters are connected, and the overall appearance is that of a casual, handwritten note.

Figure 20: An original generated sample of human-like handwriting before steganographization. This can be viewed as the cover image in steganography jargon. This showcases potential message creation of cover images and their extensibility in terms of image dimensions.

network in image recognition and classification, for feature extraction. Instead of using all seven layers, the architecture is slightly modified. Input is an $80 \times 80 \times 1$ grayscale, cropped image anchored at the top-left of the generated text image. Then, the input is passed into a convolutional layer with stride 5, 6 feature kernels, and no padding, producing activation map of dimensions $76 \times 76 \times 6$. This is passed into a max pooling layer with kernel of dimensions $2 \times 2 \times 1$, producing a $38 \times 38 \times 6$ output. This is once again passed into a similar convolutional layer (16 feature kernels now) and max pooling layer, resulting in a $17 \times 17 \times 16$ image. When flattened, it is converted to a 1×4624 feature vector, which is connected to one fully-connected layer of output size 1000. Once these features are extracted, it is passed into a LSTM (long short-term memory, a special case of a recurrent neural network) with 1003 input neurons (1000-dimension feature vector concatenated with 3-dimensional initial, null coordinate state), 900 hidden neurons, and 3 layers. A bias is included in the network, but dropout is excluded. Finally, with an output of dimensions 900×1 , this is connected to a fully-connected layer with 3 output neurons representing the initial coordinate values: x , y , eos (0 or 1 indicating end-of-sequence coordinate).

This 80×80 window is horizontally shifted and centered on the previous coordinate prediction. Then, the process is repeated to generate future coordinates. Above, a 1003-dimension feature vector is passed into the LSTM with a null, initial coordinate state. For subsequent coordinates, the coordinate state is replaced by the previous coordinate prediction. Still, this coordinate prediction is concatenated with the extracted features of the 80×80 image window. The sequence is generated until back-to-back end-of-sequence coordinates are generated. The loss function is simply measured as the distance between the true coordinate and the predicted coordinate with the L_1 distance where C' is the new, predicted coordinate and C is the old, original coordinate:

$$L_{coordinate}(C, C') = \|C' - C\|_1 = |x' - x| + |y' - y| \quad (6.1)$$

Currently, this decoding pipeline is only trained with one form of cover text, “The sky is blue”. The coordinates were passed into the steganography network with 1000, random, 310-bit data vectors to produce small perturbations of the original coordinates. In order to prevent too-fine perturbations, only every 3rd coordinate is passed into the steganography

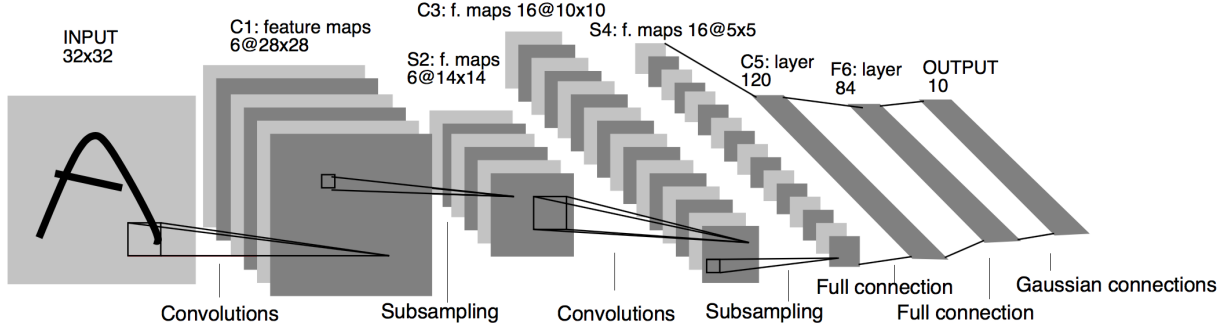


Figure 21: LeNet convolutional neural network architecture utilizing convolutional, pooling, and dense layers for image classification.

network. So, the training set for the network consists of 1000 sample container images along with 1000 perturbed coordinate sets. The network was trained for 100 epochs. Additionally, learning rate was decayed so the rate begins at 10^{-3} , is decreased to 10^{-4} after 5 epochs, decreased to 10^{-5} after 20 epochs, and finally decreased to 10^{-6} after 40 epochs.

6.3 RESULTS

The entire pipeline for both encoding and decoding data is outlined in Figure 22. The encoding process is a bit more straightforward due to simply sequentially encoding data in the coordinates and outputting them onto an image. Meanwhile, the decoding process requires an additional neural network, namely a CNN-RNN hybrid, to predict the coordinate sequence, much akin to the sequence prediction and handwriting synthesis performed in Graves' work. This decoding pipeline results in an L_1 loss of 0.0012, indicating the extracted coordinates are nearly identical to the original coordinates (e.g. they return (1.0006, 1.9994) instead of (1, 2), which is vital for the coordinate de-steganography network to extract the correct embedded data.

After some tests, two samples can be seen in Figures 23 and 24. These samples encode

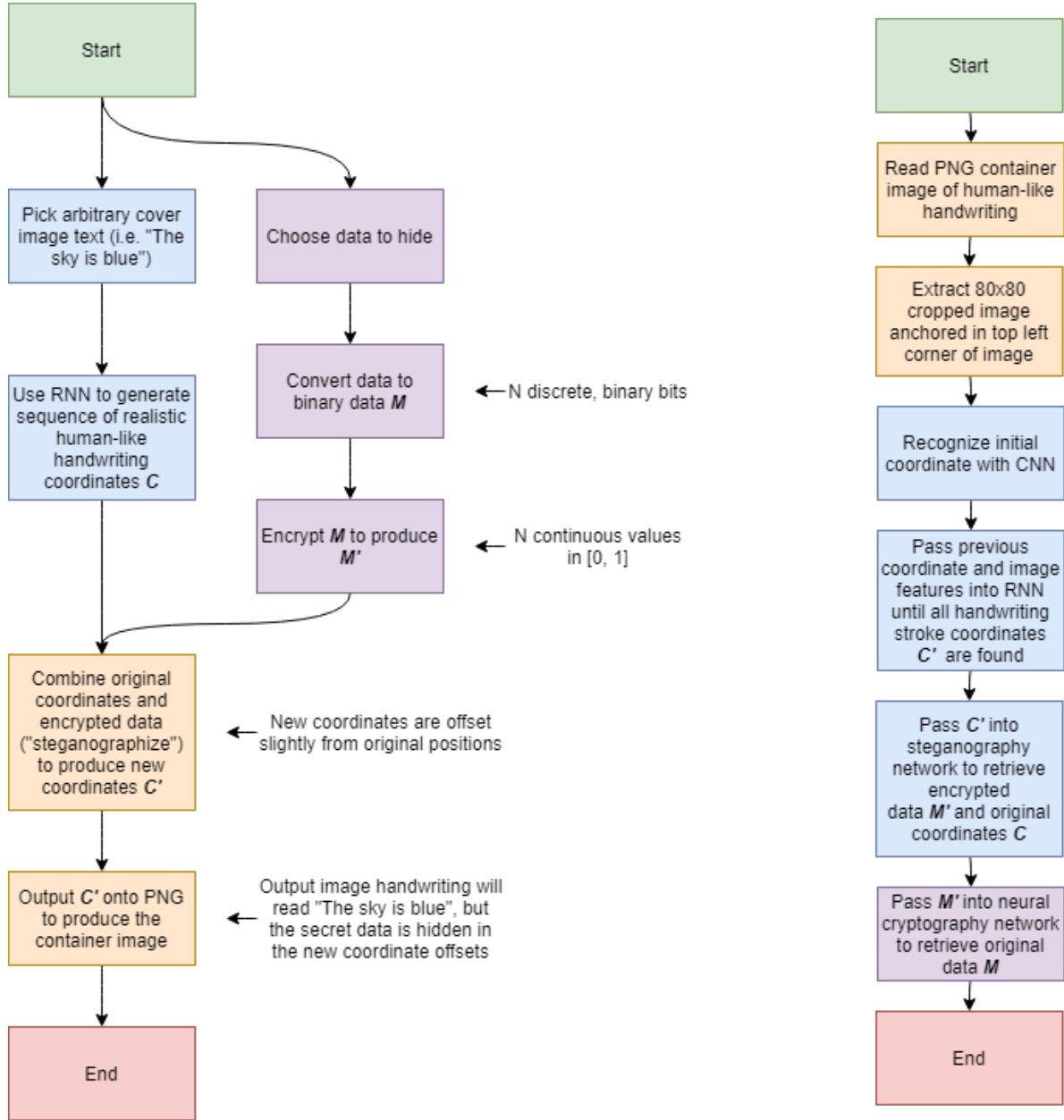


Figure 22: Left: Encoding pipeline. Right: Decoding pipeline. Blue tiles represent operations with coordinates. Purple tiles represent encryption operations. Yellow tiles represent steganography operations.

the data “The password is 123” and “Meet me on the east side at 11:03pm.” respectively. Clearly, these two images look nearly identical to the original cover image in Figure 19, but with slightly noticeable displacements in the characters. These images are a result of the encoding pipeline. When passed into the decoding pipeline, all 19 letters, or 152 bits, in “The password is 123” are recovered, as well as all 36 letters, or 288 bits, in “Meet me on the east side at 11:03pm.”.

These results are satisfactory, but could be improved for the human visual inspection task with more advanced decoding networks, allowing the encoding networks to learn more cunning and smaller perturbations. These two samples are assumed to pass the human visual inspection task as they look nearly identical to the original cover image.

These methods are not currently tested in terms of steganalysis decoding approaches. This is due to time constraints and the unconstrained container dataset size. For example, a neural network steganalysis tool reads in a pre-defined set of inputs, or size of image, which is not suitable for this problem since an arbitrary cover image can be produced of any size. However, suspicions are the ATS approach of [17] will result in similar success for two reasons. First, the underlying training dataset can be of any size, unlike the large, but defined ImageNet dataset containing 14 million images, so a network must be both generalized while also possessing the ability to detect small fluctuations in data independent of pixel intensity values. Second, the ATS approach relies on the steganography networks’ producing randomized initial seeds for weights, which is the same approach utilized both the neural cryptography and steganography networks in this paper. As such, the hope is these approaches produce similar results to the steganalysis results in [17]. However, this a path of exploration in the near future.

The image shows the handwritten text "The sky is blue" in a cursive script. The letters are black and the background is white. The text is centered horizontally and vertically within the frame.

Figure 23: A sample container image with the secret message “The password is 123” encoded in the image from Figure [19](#)

The image shows the handwritten text "The sky is blue" in a cursive script. The letters are black and the background is white. The text is centered horizontally and vertically within the frame.

Figure 24: A sample container image with the secret message “Meet me on the east side at 11:03pm.” encoded in the image from Figure [19](#)

7.0 CONCLUSIONS

With these results, we have shown a fully integrated steganographic pipeline capable of learning sequential representations of data with low reconstruction error. As such, this can easily be put into practice with some use. With that being said, there is much room for improvement in terms of generalizability and exploration of this new direction of steganography.

7.1 DISCUSSION

This pipeline showcases a complete encoding and decoding approach to steganography unseen before. Although it follows the recent 5-year trend of neural steganography, it expands on current approaches by *modifying* data, rather than embedding data. Furthermore, although a small embedding capacity was achieved, the hope is the tradeoff in data reconstruction and steganographic success pays off. The embedding capacity is a small issue that is hoped to be fixed with future advancements in neural networks or further exploration of network architectures.

With this pipeline, initially designed to embed 8-bit ASCII characters, UTF-8 and UTF-16 characters should be just as easily emplaced into the image by splitting up the bits into chunks of 2 and combining them with coordinates, assuming the data is successfully retrieved. Furthermore, an entire image can also be embedded into the cover if the 8-bit pixels are broken into chunks of 2 bits. So, this pipeline should be able to work with both image and non-image data. This showcases the flexibility on the restriction for input data to have spatial structure, as required in typical convolutional neural networks and convolutional

autoencoders.

This work has shown a clear, new direction for the field of steganography. In contrast with traditional *embedding* approaches, the proposed pipeline *modifies* the cover image’s characteristics and high-level details. With these improvements, it is impossible for the human eye to spot differences in normal and steganographic images. In addition, this work has shown to fix the issues with sparsity as mentioned when discussing [1] and [17]. Although steganalysis tests were not performed, they were thoroughly discussed in Section 6.3 in the hopes that tests in the near future result in a similar detection rate of 50% (random guessing) when an adversary has access to container images.

7.2 LIMITATIONS

Although some results are produced, there are a few limitations to be discussed in this work. For example, the steganography network only embeds two bits of data per coordinate. As such, when working with a limited number of coordinates in the generated handwriting, this is a bottleneck in terms of the amount of secret information capable of fitting into smaller cover images. Second, the neural cryptography is pre-trained to embed chunks of 128 bits of secret data at a time. If the same key is utilized to encrypt each chunk, repetition is bound to occur in the encrypted chunks. Thus, an adversary may notice these repetitions and reverse engineer the secret information. For example, if the messages “THE ANIMAL IS SHORT” is passed into a Caesar cipher, the resultant message after a shift of 3 letters is “WKH DQLPDO LV VKRUW”. One observes every repeated A is converted to “D” and every S is converted to “V”. This is referred to as an electronic code book [99]. One solution to this is CBC (Cipher Block Chaining), which is discussed in the future work [99]. Third, the current implementations only utilize one style of handwriting and one bias level with a fixed cover message (“The sky is blue”). As such, adversaries with this knowledge should be easily capable of training against this fixed dataset to recover the secret data. Finally, when this pipeline is put into practice, for example on a social media site, compression techniques may impact the dimensions and locations of data in the image. With significant

downscaling, a significant amount of information is lost in terms of data placement, but the general integrity remains. For example, we know where the coordinates lie with respect to each other, but rescaling the image back up to its original size results in blurred pixels, which is vital to the success of this approach. If only a few pixels are blurred, the approach is robust, but a significant chunk of blurred pixels results in incorrect displacement predictions for the decoding pipeline. As such, there are a few issues to fix in the future. Solutions to these limitations are briefly discussed in the future work section below.

7.3 PROBLEM GENERALIZABILITY

Using handwriting as a form of steganography was always a toy problem in this domain to showcase verification of the idea working. However, this idea seems to be transferrable to other domains. For example, steganography typically deals with images or audio data, but can it be applied to a video where the optical flow dynamics of a video are modified to encode data? Although ambitious, simpler characteristics might be more feasible. For example, colors of objects may be an approachable start. Given an image of people, can the colors of the shirts, pants, shoes, etc. be modified in terms of colors and patterns to embed high-dimensional data? If there are 1024 different colors and patterns for a given article of clothing, one might be able to embed $\log_2 1024 = 10$ bits of information per article of clothing. As such, this work seems feasible and less susceptible to being spotted by both the naked human eye and without a doubt robust to machine inspection. Therefore, it should easily pass the two necessary tasks in steganography to be considered a success.

7.4 FUTURE WORK

This work can be expanded through several different avenues across the entire pipeline: encryption, embedding capacity, generalizing the problem, and steganographization. Formulations of ideas can be seen below:

First, encryption can be explored a bit further. As already discussed in this chapter, results in terms of encryption were notable. A 128-bit vector of data was able to be successfully and securely encrypted while being robust against attackers. However, the Eve and Dave, the adversaries, were trained with fairly shallow networks. Further exploration into them as deep networks vs. Alice and Bob being shallow, or all agents being deep neural networks, might be useful paths of exploration in terms of enhancing the final barrier of security in this pipeline, but also for verification of the success of neural cryptography. Finally, new data/key splits may be explored further to examine varying success with key sizes and data/key ratios. For example, the neural cryptography algorithms trained in this paper used 128-bit data and a 128-bit key to produce a 256-bit input vector, resulting in a 50/50 split between the data and key. However, are all 128 bits necessary? As discussed already, well-known, proven security methods, AES and RSA, commonly use 256-bit and 4096-bit keys to secure the data. Further along in the pipeline, we learn this steganographic approach uses a symmetric key, so does a 256-bit key increase security and robustness of the algorithms compared to the 128-bit key currently used?

Second, the embedding capacity is fairly low per coordinate. As discussed, 2-LSB can produce a 0.250 bpp embedding, but is easily detectable. [1] uses 1.0 bpp, but each decoded byte of data is off by an average intensity of 3.0, which essentially transforms non-image data into a jumbled mess. Meanwhile, [17] uses 0.203 bpp and produces an error of 10^{-5} , but suspicions are the images are detectable by steganalysis tools if ‘1 of n ’ samples were to be given to a network and the network had to decide which image was steganographic. In this paper, a 25% embedding ratio per pixel is created, but not every pixel in the image contains embedded information. With that being said, with further exploration, if a neural network is found to embed 8 bits per coordinate, entire ASCII characters can be embedded at a time. This reduces the risk of data reconstruction since the message “The cassword is 123” can be autocorrected to “The password is 123” through breakthroughs in natural language processing [100]. So, can a neural network successfully learn displacements for coordinates for data with more than 2 bits?

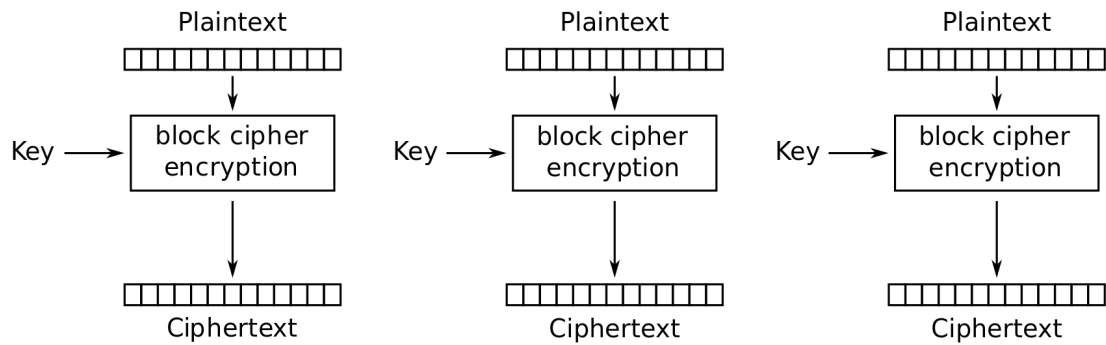
Third, the toy problem of handwriting generation worked as a toy problem. It showcased steganography is more than embedding information into an image, but rather it provides

insight into new avenues in steganography. Other than handwriting, can this approach be applied to other domains, such as modifying the picture of a cat to have different colored eyes or longer whiskers? Is it possible to generalize this problem to all domains of images to pick out the most important aspects of an image, and modify their color or appearance with the use of specified bits, and appear even less susceptible to attackers?

Next, as discussed above, the current neural cryptography approaches use ECB (Electronic Codebook), which utilizes an identical key to encrypt chunked data, resulting in duplicates of similar bits of data, such as in the aforementioned Caesar cipher example. Cipher block chaining (CBC) overcomes this limitation by bitwise XORing the previously encrypted ciphertext with the next chunk’s plaintext, resulting in a unique combination of data. These two schemes can be observed in Figures 25 and 26. This approach can be improved upon by utilizing a recurrent neural network to learn some hash or pointwise operation to transform the data in sequential chunks, converting the current ECB neural cryptography network to a CBC neural cryptography network.

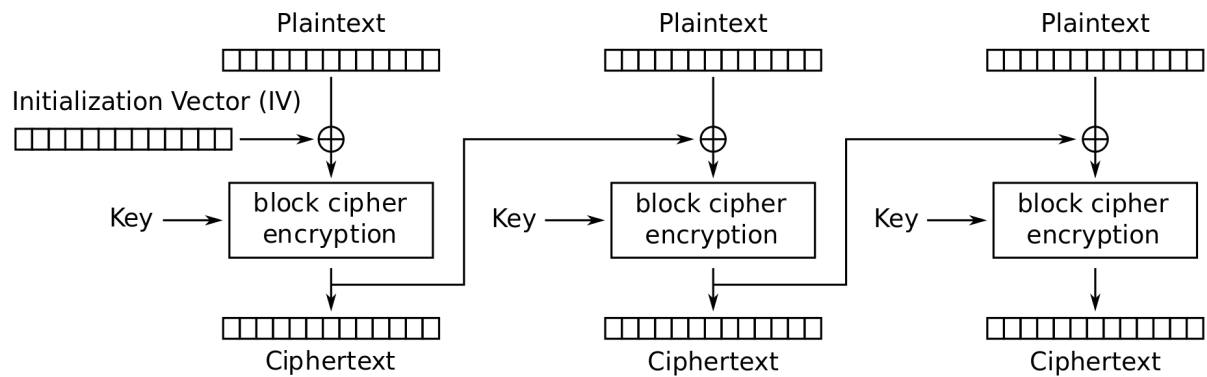
Also, this work can be extended to another form of generative model, a sequence GAN [102]. A sequence GAN is similar to a vanilla GAN where the primary goal is to train a generator network to create fake dataset samples, reflective of the training dataset, with the hopes of maximizing the error of a discriminator network, which classifies each generated sample as real or fake. With a sequence GAN, this work can be further abstracted to generate a realistic sequence of coordinates which appear similar to realistic human handwriting, but combined with some form of secret data.

In terms of future evaluation, several metrics come to mind. The first is to simply train a binary classifier on the training dataset of container images against the underlying dataset of cover images of generated handwriting. This classifier would simply predict whether a given image is steganographic or not. Second, the approach in [17] can be utilized, where five classifiers are trained on five independent training sets of container and cover images, and then these five classifiers test on an independent, sixth set of data. Thus, the adversaries have access to the network and its outputs, but without complete access to the training data, the adversaries might fail. In terms of embedding ratio metrics, the ratios discussed throughout this paper do not properly assess this new, proposed approach. Those ratios



Electronic Codebook (ECB) mode encryption

Figure 25: An electronic codebook encryption scheme for chunked data as seen in [101]



Cipher Block Chaining (CBC) mode encryption

Figure 26: A cipher block chaining encryption scheme for chunked data as seen in [101]

simply take into account the number of secret bits of data and the number of total bits in the cover data or image. This paper attempts to overcome the sparsity issues in those papers, so a new metric, not based purely on density of the entire image, but rather density across the utilized portions of the image. As such, one appropriate metric might be to divide the number secret data bits by the number of utilized image features (i.e. coordinates), which is $314/157 = 2$ in this case, then divide that by 8 (due to 8 bits per pixel), resulting in a 25% embedding ratio. Although the metric accounts for the image sparsity, new metrics combining both sparsity and non-sparsity should be explored.

Finally, the field of steganography is captivating. With a sudden lull in the 1990s and 2000s, the 2010s have produced fascinating results in the field with the emergence of convolutional neural networks capable of efficiently representing spatial structure in images. However, as the field of security advances, so does the state of steganography. Steganography needs to become more advanced, which requires more advanced pipelines. With the pipeline proposed in this paper, can a generalized, streamlined pipeline be generated to steganographize any image in any domain with any encryption scheme, as if the parts were swappable?

BIBLIOGRAPHY

- [1] Shumeet Baluja. Hiding images in plain sight: Deep steganography. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2069–2079. Curran Associates, Inc., 2017.
- [2] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn. Information hiding-a survey. *Proceedings of the IEEE*, 87(7):1062–1078, July 1999.
- [3] Walter Bender, Daniel Gruhl, and Norishige Morimoto. Techniques for data hiding. In *Storage and Retrieval for Image and Video Databases*, 1995.
- [4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.
- [5] ProZ. English translation: stagnos - covered writing, 2002.
- [6] ProZ. English translation: graphien - write, 2002.
- [7] G. Sugandhi and C. P. Subha. Efficient steganography using least significant bit and encryption technique. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–6, Jan 2016.
- [8] National Institute of Standards and Technology (NIST). Announcing the advanced encryption standard (aes), 2001.
- [9] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.
- [10] Ekta Walia and Payal Jain. An Analysis of LSB & DCT based Steganography. 2010.
- [11] Deepak Singla and Rupali Syal. Data security using LSB & DCT steganography in images. 03 2019.
- [12] Dave Marshall. The discrete cosine transform (dct), 2001.
- [13] Stanford Vision Lab. Imagenet, 2016.

- [14] J. Deng, W. Dong, R. Socher, L. Li, and and. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.
- [15] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- [16] Neil F. Johnson. Steganography software, 2012.
- [17] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. Hidden: Hiding data with deep networks. *CoRR*, abs/1807.09937, 2018.
- [18] Daniel Lerch-Hostalot and David Megías. Unsupervised steganalysis based on artificial training sets. *CoRR*, abs/1703.00796, 2017.
- [19] ICFHR 2018. Icfhr: International conference on frontiers in handwriting recognition, 2018.
- [20] dblp. International conference on frontiers in handwriting recognition, 2019.
- [21] C. Y. Suen. First international workshop on frontiers in handwriting recognition (iwfhr-1), 2010.
- [22] Vedran Vukotic, Silvia-Laura Pintea, Christian Raymond, Guillaume Gravier, and Jan C. van Gemert. One-step time-dependent future video frame prediction with a convolutional encoder-decoder neural network. *CoRR*, abs/1702.04125, 2017.
- [23] William Lotter, Gabriel Kreiman, and David D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. *CoRR*, abs/1605.08104, 2016.
- [24] Yunbo Wang, Mingsheng Long, Jianmin Wang, Zhifeng Gao, and Philip S. Yu. Predrnn: Recurrent neural networks for predictive learning using spatiotemporal lstms. In *NIPS*, 2017.
- [25] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.
- [26] Liqian Ma, Xu Jia, Qianru Sun, Bernt Schiele, Tinne Tuytelaars, and Luc Van Gool. Pose guided person image generation. *CoRR*, abs/1705.09368, 2017.
- [27] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.
- [28] Parnia Bahar, Christopher Brix, and Hermann Ney. Towards two-dimensional sequence to sequence model in neural machine translation. *CoRR*, abs/1810.03975, 2018.

- [29] Qi Wu, Peng Wang, Chunhua Shen, Anton van den Hengel, and Anthony R. Dick. Ask me anything: Free-form visual question answering based on knowledge from external sources. *CoRR*, abs/1511.06973, 2015.
- [30] B. L. D. Bezerra, C. Zanchettin, and V. Braga de Andrade. A hybrid rnn model for cursive offline handwriting recognition. In *2012 Brazilian Symposium on Neural Networks*, pages 113–118, Oct 2012.
- [31] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, May 2009.
- [32] Bruno Stuner, Clément Chatelain, and Thierry Paquet. Cohort of LSTM and lexicon verification for handwriting recognition with gigantic lexicon. *CoRR*, abs/1612.07528, 2016.
- [33] Cheng-Lin Liu, Kazuki Nakashima, Hiroshi Sako, and Hiromichi Fujisawa. Handwritten digit recognition: Benchmarking of state-of-the-art techniques. *Pattern Recognition*, 36:2271–2285, 10 2003.
- [34] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 545–552. Curran Associates, Inc., 2009.
- [35] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.
- [36] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.
- [37] JURGEN SCHMIDHUBER. *Netzwerkarchitekturen, Zielfunktionen und Kettenregel (Network architectures, objective functions, and chain rule)*. PhD thesis, Technische Universitt Mnchen, <http://people.idsia.ch/juergen/onlinepub.html>, 1993.
- [38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.
- [39] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.
- [40] Serena Yeung Fei-Fei Li, Justin Johnson. Lecture 10: Recurrent neural networks, 2017.
- [41] Saurabh Rathor. Simple rnn vs gru vs lstm : Difference lies in more flexible control, 2018.

- [42] Sam Greydanus. Scribe: realistic handwriting with tensorflow, 2016.
- [43] hardmaru. Handwriting generation demo in tensorflow, 2015.
- [44] Shan Carter, David Ha, Ian Johnson, and Chris Olah. Experiments in handwriting with a neural network. *Distill*, 2016.
- [45] lirnli. Notes: Fake handwriting generation with pytorch, 2017.
- [46] Caglar Gulcehre. Deep learning...moving beyond shallow machine learning since 2006!, 2014.
- [47] Snowkylin Lazarus. Handwriting generation by rnn with tensorflow, based on "generating sequences with recurrent neural networks" by alex graves, 2017.
- [48] Grzegorz Opoka. Implementation of handwriting generation with use of recurrent neural networks in tensorflow. based on alex graves paper (<https://arxiv.org/abs/1308.0850>), 2018.
- [49] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.
- [50] Rjean Plamondon. Looking at handwriting generation from a velocity control perspective. *Acta Psychologica*, 82(1):89 – 101, 1993.
- [51] Emre Aksan, Fabrizio Pece, and Otmar Hilliges. Deepwriting: Making digital ink editable via deep generative modeling. *CoRR*, abs/1801.08379, 2018.
- [52] Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction. *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, 2:28–31 Vol.2, 2004.
- [53] Christopher M. Bishop. Mixture density networks. 01 1994.
- [54] FKI: Research Group on Computer Vision and University of Bern Artificial Intelligence INF. Iam on-line handwriting database, 2019.
- [55] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.
- [56] Yann LeCun. The mnist database of handwritten digits, 2012.
- [57] Patrick J. Grother. Nist handprinted forms and characters database, 2018.
- [58] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373, 2017.

- [59] Jonathan Tapson Gregory Cohen, Saeed Afshar and Andre van Schaik. The emnist dataset, 2018.
- [60] Marcus Liwicki and Horst Bunke. Iam-ondb - an on-line english sentence database acquired from handwritten text on a whiteboard. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition, ICDAR '05*, pages 956–961, Washington, DC, USA, 2005. IEEE Computer Society.
- [61] Luidia Inc. ebeam interactive suite 3, 2019.
- [62] Sean Vasquez. Handwriting synthesis with rnns, 2018.
- [63] William August Kotas. A brief history of cryptography. Undergraduate honors thesis, University of Tennessee - Knoxville.
- [64] Learn Cryptography. Caesar cipher, 2019.
- [65] May Aung and Myat Wai. Study on symmetric and asymmetric cryptographic techniques. 02 2015.
- [66] Gustavus J. Simmons. Advanced encryption standard, 2014.
- [67] M. E. Smid and D. K. Branstad. Data encryption standard: past and future. *Proceedings of the IEEE*, 76(5):550–559, May 1988.
- [68] Evgeny Milanov. The rsa algorithm. 2009.
- [69] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, October 1997.
- [70] Dan Boneh et al. Twenty years of attacks on the rsa cryptosystem. *Notices of the AMS*, 46(2):203–213.
- [71] Ido Kanter, Wolfgang Kinzel, and Eran Kanter. Secure exchange of information by synchronization of neural networks. *EPL (Europhysics Letters)*, 57, 02 2002.
- [72] Nankun Mu and Xiaofeng Liao. An approach for designing neural cryptography. In Chengan Guo, Zeng-Guang Hou, and Zhigang Zeng, editors, *Advances in Neural Networks – ISNN 2013*, pages 99–108, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [73] Oscar Mauricio Reyes and Karl-Heinz Zimmermann. Permutation parity machines for neural cryptography. *Phys. Rev. E*, 81:066117, Jun 2010.
- [74] Alexander Klimov, Anton Mityagin, and Adi Shamir. Analysis of neural cryptography. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, pages 288–298, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
- [75] Martín Abadi and David G. Andersen. Learning to protect communications with adversarial neural cryptography. *CoRR*, abs/1610.06918, 2016.

- [76] Dana H. Ballard. Modular learning in neural networks. In *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, AAAI'87, pages 279–284. AAAI Press, 1987.
- [77] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.
- [78] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010.
- [79] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313 5786:504–7, 2006.
- [80] Murilo Coutinho, Robson de Oliveira Albuquerque, Fábio Borges, L. Javier García-Villalba, and Tai-Hoon Kim. Learning perfectly secure cryptography to protect communications with adversarial neural cryptography. In *Sensors*, 2018.
- [81] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [82] TensorFlow. Gradientdescentoptimizer, 2019.
- [83] S.S. Haykin. *Neural Networks: A Comprehensive Foundation*. International edition. Prentice Hall, 1999.
- [84] EDUCBA. Cryptography vs encryption, 2019.
- [85] Andreas Litke Kostas Zotos. Cryptography and encryption, 2005.
- [86] Johannes Trithemius. *Steganographia*. 1499.
- [87] Mansi S. Subhedar and Vijay H. Mankar. Current status and key issues in image steganography: A survey. *Computer Science Review*, 13-14:95 – 113, 2014.
- [88] F.Y. Shih. *Digital Watermarking and Steganography: Fundamentals and Techniques, Second Edition*. CRC Press, 2017.
- [89] I. Cox, M. Miller, J. Bloom, J. Fridrich, and T. Kalker. *Digital Watermarking and Steganography*. The Morgan Kaufmann Series in Multimedia Information and Systems. Elsevier Science, 2007.
- [90] C.N. Yang, C.C. Lin, and C.C. Chang. *Steganography and Watermarking*. Privacy and Identity Protection Series. Nova Science Publishers, Incorporated, 2014.

- [91] Jonathan Cummins, Patrick Diskin, Samuel Lau, and Robert Parlett. Steganography and digital watermarking. 01 2004.
- [92] Derrick Grover. Steganography for identifying ownership of copyright. *Computer Law Security Report*, 14:121–122, 03 1998.
- [93] Black Slash. How to hide secret data inside an image or audio file in seconds, 2018.
- [94] b3dk7. Stegexpose, 2015.
- [95] Alfonso Muoz. Stegsecret. a simple steganalysis tool ;), 2007.
- [96] Binghamton University DDE Lab. Feature extractors for steganalysis, 2018.
- [97] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.
- [98] Yann LeCun et al. Lenet-5, convolutional neural networks.
- [99] Steven M. Bellovin. Modes of operation, 2009.
- [100] Neha Gupta and Pratistha Mathur. Spell checking techniques in nlp: a survey. 2012.
- [101] Scott Fazackerley, Steven M. McAvoy, and Ramon Lawrence. Gpu accelerated aes-cbc for database applications. pages 873–878, 03 2012.
- [102] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.